HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

BACHELORARBEIT

Frau
Julia Schuster

Untersuchung von Optimierungsverfahren der ein- und multikriteriellen Optimierung mit diskreten Parametern für Anwendungen in der Antriebsstrangauslegung

2010

Fakultät: **Mathematik/Naturwissenschaften/Informatik**

Bachelorarbeit

Untersuchung von Optimierungsverfahren der ein- und multikriteriellen Optimierung mit diskreten Parametern für Anwendungen in der Antriebsstrangauslegung

Autor:
Julia Schuster

Studiengang:
Angewandte Mathematik

Seminargruppe:
MA07w1-B

Erstprüfer:
Prof. Dr. rer. nat. Regina Fischer

Zweitprüfer:
Dipl. Math. (FH) Steffen Kux

Mittweida, September 2010

Bibliographische Angaben

Schuster, Julia: Untersuchung von Optimierungsverfahren der ein- und multikriteriellen Optimierung mit diskreten Parametern für Anwendungen in der Antriebsstrangauslegung, 66 Seiten, 20 Abbildungen, Hochschule Mittweida (FH), Fakultät Mathematik/Naturwissenschaften/Informatik

Bachelorarbeit, 2010

Referat

Da bei technischen Optimierungsaufgaben oft diskrete Parameter wie Stückzahlen oder die Anzahl der Zähne eines Zahnrades, welche nur ganzzahlige Werte annehmen können, eine Rolle spielen, können Optimierungsverfahren, welche für stetige Parameter entwickelt wurden, nicht ohne Weiteres für diese Aufgaben eingesetzt werden. Das Ziel dieser Arbeit ist es, eine Diskretisierung ausgewählter Optimierungsverfahren zu entwickeln und diese programmtechnisch umzusetzen. Anschließend sollen die diskretisierten Verfahren auf ihre Einsetzbarkeit getestet und die Ergebnisse miteinander verglichen werden, so dass eine Empfehlung zur Anwendung der Optimierungsverfahren gegeben werden kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Abkürzungsverzeichnis

DOA	Diskrete Optimierungsaufgabe
EOA	Einkriterielle Optimierungsaufgabe
KOA	Kombinatorische Optimierungsaufgabe
MOA	Multikriterielle Optimierungsaufgabe
OA	Optimierungsaufgabe
BBM	Branch-and-Bound-Methode
BCM	Branch-and-Cut-Methode
DiRect	Dividing-Rectangles
EMMOA	Enhanced Modelbased Multiobjective Optimization Algorithm
MAES	Model Assisted Evolutionary Strategy
OGA	Ordinary Genetic Algorithm
PSO	Partikelschwarmoptimierung
UNO	Universal Optimizer

Symbolverzeichnis

$\mathbb{N}$	Menge der natürlichen Zahlen einschließlich 0
$\mathbb{R}$	Menge der reellen Zahlen
x^*	optimale Lösung einer Optimierungsaufgabe
X^*	Menge aller optimalen Lösungen einer Optimierungsaufgabe
X_D	diskrete Menge
x_L	untere Grenze des Parameters x
x_U	obere Grenze des Parameters x
s	Schrittweite einer Diskretisierung
i	Standübersetzung
p	Anzahl der Planetenräder
z_S	Zähnezahl des Sonnenrades
z_P	Zähnezahl der Planetenräder
z_H	Zähnezahl des Hohlrades

1 Einleitung

1.1 Motivation und Aufgabenstellung

Bei praktischen Optimierungsproblemen kommt es oft vor, dass die Parameter bestimmte Diskretheitsforderungen erfüllen müssen. Beispielsweise können Zahnräder nur eine natürliche Anzahl an Zähnen haben oder Kosten für ein Produkt nur reelle Werte mit zwei Nachkommastellen annehmen. Die meisten Optimierungsverfahren können zunächst jedoch nur für reelle Parameter genutzt werden und ein nachträgliches Runden der stetigen Lösung auf diskrete Werte führt meist zu unbrauchbaren Ergebnissen. Daher ist es nötig, die Verfahren der stetigen Optimierung zu diskretisieren.

Ziel dieser Arbeit ist es, für vier ausgewählte Optimierungsverfahren eine geeignete Diskretisierung zu finden, welche für praktische Aufgaben genutzt werden kann. Dazu müssen zunächst die grundlegenden Begriffe der Optimierung und die betrachteten Optimierungsverfahren erläutert werden. Anschließend können die Verfahren diskretisiert werden, wobei auf die Besonderheiten der einzelnen Verfahren zu achten ist. Zuletzt muss überprüft werden, ob eine Diskretisierung für verschiedene Aufgaben exakt realisiert werden kann und ob die Resultate den gewünschten Ergebnissen entsprechen.

1.2 Vorstellung IAV GmbH

Die Ingenieurgesellschaft Auto und Verkehr, kurz IAV GmbH, ist ein weltweit agierendes Entwicklungsunternehmen für die Automobil- und Zulieferindustrie. Mit mehr als 3 000 Mitarbeitern in den Bereichen der Antriebsstrang-, Elektronik- und Fahrzeugentwicklung ist das 1983 in Berlin gegründete Unternehmen nach ATZ [25] das viertgrößte Engineering-Unternehmen weltweit. Im Firmenprofil [15] wird die IAV GmbH als "Bindeglied zwischen Idee und Serienreife" bezeichnet, da sie die Entwicklung neuer Produkte vom Konzept über die Berechnung und Konstruktion bis hin zum Prototypenbau begleitet. Dabei werden die vier Arbeitsschwerpunkte Powertrain-Mechatronik, Powertrain-Entwicklung, Fahrzeug-Entwicklung und Fahrzeug-Elektronik von erfahrenen Ingenieuren mit höchster technischer Ausstattung zielgerichtet und kostenorientiert bearbeitet. Zu den langjährigen Kunden der IAV GmbH zählen nahezu alle Automobilhersteller und ihre Systemzulieferer (vgl. [15]).
Der Standort Chemnitz der IAV GmbH mit aktuell ca. 500 Angestellten wurde 1993 als IAV Motor GmbH Chemnitz gegründet und umfasst die Geschäftsbereiche Development-Powertrain, Fahrzeug-

Elektronik und Powertrain-Mechatronik. Neben der Entwicklung und Fertigung von Motoren und Elektroniksystemen wird von der Development-Powertrain-Abteilung *Prozesse* das Softwareprogramm *EngineeringToolbox* bereitgestellt, welches von der IAV GmbH entwickelt wurde und beständig erweitert wird. Mit der *EngineeringToolbox* können u.a. Optimierungsaufgaben gelöst und Toleranzsimulationen durchgeführt werden. Außerdem bietet diese Software einen umfassenden Wissensspeicher zu vielen technischen Entwicklungen und Produkten. Eine Beschreibung der *EngineeringToolbox* findet sich im Anhang A dieser Arbeit. Die Optimierungsmöglichkeiten der *EngineeringToolbox* sollten nun im Rahmen dieser Arbeit für diskrete Optimierungsprobleme erweitert werden.

1.3 Gliederung der Bachelorarbeit

Zu Beginn dieser Arbeit werden die grundlegenden Begriffe der Optimierung geklärt. Es wird dabei sowohl auf die allgemeine Problemstellung einer Optimierungsaufgabe als auch speziell auf ein- und multikriterielle sowie diskrete Optimierungsaufgaben eingegangen. Außerdem werden erste Lösungsansätze für spezielle Optimierungsprobleme beschrieben. Im Anschluss werden die drei Optimierungsverfahren *Dividing-Rectangles*, *Nelder-Mead* und *Partikelschwarmoptimierung* sowie die *Ersatzmodell-gestützten Verfahren* erläutert. Dabei wird der allgemeine Verfahrensablauf beschrieben sowie Vor- und Nachteile dieser Methoden aufgezeigt. Das nachfolgende Kapitel beschäftigt sich mit der Diskretisierung dieser Optimierungsverfahren. Es wird zunächst auf das allgemeine Prinzip der Diskretisierung eingegangen und anschließend werden die speziellen Verfahren näher beschrieben. Im fünften Kapitel werden die Testergebnisse der diskretisierten Optimierungsverfahren dargelegt und mit deren Hilfe eine Einschätzung der Diskretisierung gegeben. Das letzte Kapitel liefert eine Zusammenfassung dieser Arbeit und einen Ausblick für weiterführende Aufgaben.

2 Grundlagen der Optimierung

In diesem Kapitel werden die grundlegenden Begriffe und Definitionen der Optimierung geklärt. Es wird zunächst eine allgemeine Aufgabenstellung formuliert, um dann auf die Unterteilung in ein- und multikriterielle Optimierungsaufgaben einzugehen. Der Abschnitt 2.4 widmet sich speziell der diskreten Optimierung.
Die nachfolgenden Bezeichnungen und Definitionen wurden aus [1], [11], [12] und [13] entnommen.

2.1 Allgemeine Problemstellung

Eine Optimierungsaufgabe (OA) lässt sich allgemein wie folgt definieren:

Definition 2.1: Optimierungsaufgabe
Sei X eine nichtleere Teilmenge des $\mathbb{R}^n$ und $f : X \to \mathbb{R}^m$ die zu optimierende Funktion. Dann nennt man

$$\begin{aligned} f(x) &\to opt \\ x &\in X \subseteq \mathbb{R}^n \end{aligned} \tag{2.1}$$

eine *(allgemeine) Optimierungsaufgabe*.
Die Funktion f wird dabei als *Zielfunktion* bezeichnet, die Menge X heißt *zulässige Menge* oder *zulässiger Bereich*. Die Elemente x nennt man *zulässige Punkte* bzw. *zulässige Lösungen*. Die Komponenten von x werden häufig als *Variablen*, in der Technik auch als *Entwurfsparameter* oder kurz *Parameter* bezeichnet.
Den Wertebereich der Zielfunktion nennt man auch *Zielbereich* oder *Suchraum*.

Als Optimierungsrichtung kann eine Minimierung oder eine Maximierung von f gewählt werden. In dieser Arbeit werden jedoch o.B.d.A. nur Minimierungsprobleme betrachtet, da sich jede Maximierung von f durch eine Minimierung von $-f$ ausdrücken lässt. Es werden folglich nur Optimierungsaufgaben der Form

$$\begin{aligned} f(x) &\to min \\ x &\in X \subseteq \mathbb{R}^n \end{aligned} \tag{2.2}$$

betrachtet. Eine andere Schreibweise dieses Problems, welche häufig in der Literatur verwendet wird, lautet:

$$\min_{x \in X \subseteq \mathbb{R}^n} f(x) \tag{2.3}$$

Der zulässige Bereich X wird meist durch Nebenbedingungen bzw. *Restriktionen* aus der Gesamtmenge $\mathbb{R}^n$ bestimmt. Diese Restriktionen bestehen aus Gleichungen, Ungleichungen oder Operatorgleichungen. Zusätzlich kann die zulässige Menge durch Ganzzahligkeits- oder andere Forderungen an x beschränkt werden. Gilt $X = \mathbb{R}^n$, so spricht man von einer *unrestringierten Optimierungsaufgabe*. Ist jedoch $X \subset \mathbb{R}$, so nennt man diese Optimierung *restringiert*. Eine restringierte OA mit k Ungleichungen $g_i(x) : \mathbb{R}^n \to \mathbb{R}$ und l Gleichungen $h_j(x) : \mathbb{R}^n \to \mathbb{R}$ lässt sich ausführlich schreiben als:

$$\begin{aligned} f(x) &\to min \\ g_i(x) &\leq 0, \quad i = 1, 2, ..., k \\ h_j(x) &= 0, \quad j = 1, 2, ..., l \\ x &\in \mathbb{R}^n \end{aligned} \tag{2.4}$$

Bemerkungen:

- Man kann sich o.B.d.A. auf Ungleichungen ≤ 0 beschränken, da sich jede Ungleichung in eine solche Form überführen lässt.
- Gilt für ein $x^\circ \in X$ $g_i(x^\circ) = 0$, so nennt man diese Ungleichung *in* x° *aktiv*.
- Die Ungleichungen und Gleichungen lassen sich wie folgt zu Vektorfunktionen zusammenfassen:

$$g_i(x) \leq 0, \forall i \;\Leftrightarrow\; g(x) \leq 0^1 \text{ und } h_j(x) = 0, \forall j \;\Leftrightarrow\; h(x) = 0^1$$
$$\text{mit } g(x) : \mathbb{R}^n \to \mathbb{R}^k \text{ und } h(x) : \mathbb{R}^n \to \mathbb{R}^l$$

- Wird eine Komponente x_i von zwei Ungleichungen der Art $x_i^L \leq x_i \leq x_i^U$ beschränkt, so nennt man diese Restriktionen *Box-Restriktionen*.

Aus der Definition (2.4) einer OA lassen sich verschiedene Spezialfälle ableiten, die hier jedoch nur genannt werden sollen und später nicht weiter betrachtet werden, da die allgemeinen Lösungsverfahren auch auf diese Spezialfälle angewandt werden können. Solche Spezialfälle ergeben sich z.B. aus der Linearität der Nebenbedingungen – man nennt diese Optimierungsaufgaben dann *linear restringiert*. Ist die Zielfunktion zusätzlich linear oder quadratisch, so spricht man von einer *linearen* bzw. *quadratischen Optimierung*. Außerdem gibt es noch die *konvexe Optimierung*, bei der sowohl die Zielfunktion als auch die Nebenbedingungen konvex sind.
Eine weitere Klasse der Optimierungsaufgaben bilden die *diskreten Optimierungsaufgaben*, bei denen der zulässige Bereich nur aus diskreten, nicht zusammenhängenden Werten besteht. So könnte die zulässige Menge z.B. nur die natürlichen, die rationalen Zahlen oder Teilmengen dieser Zahlen umfassen. Die diskrete Optimierung wird in Abschnitt 2.4 näher betrachtet.

[1] In diesen beiden Fällen steht 0 für den Nullvektor.

2.2 Einkriterielle Optimierungsaufgaben

Eine Optimierungsaufgabe nennt man *einkriteriell*, wenn der Wertebereich der Zielfunktion nur die reellen Zahlen umfasst, d.h. $f : X \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$. Wir erhalten somit die OA:

$$\begin{aligned} f(x) &\rightarrow min \\ g(x) &\leq 0 \\ h(x) &= 0 \\ x &\in \mathbb{R}^n \end{aligned} \tag{2.5}$$

Für einkriterielle Optimierungsaufgaben (EOA) lassen sich leicht die folgenden Begriffe definieren:

Definition 2.2: Minimalstelle, Minimalwert
Es sei $x^* \in X$ eine zulässige Lösung von (2.5) und $U(x^*) \subseteq X$ eine Umgebung von x^* in X. Wenn

$$f(x^*) \leq f(x),\ \forall x \in U(x^*)$$

gilt, dann heißt x^* *(lokale) Minimalstelle* und $f(x^*)$ *(lokaler) Minimalwert* oder kurz *Minimum*. Für $U(x^*) = X$ ist x^* eine *globale Minimalstelle* und $f(x^*)$ ein *globales Minimum*. Gilt sogar die strenge kleiner-als-Relation, so spricht man von einem *strikten (lokalen/globalen) Minimum*.
Analoge Begriffe können für Maximierungsaufgaben definiert werden.

Viele Verfahren zur Bestimmung (lokaler) Optima benutzen notwendige und hinreichende Optimalitätskriterien. Zu den bekanntesten notwendigen Kriterien gehören die Fritz John-Bedingungen und die Karush-Kuhn-Tucker-Bedingungen mit geeigneten Regularitätsbedingungen. Ein hinreichendes Optimalitätskriterium bieten die Sattelpunktsbedingungen, aus denen sich wiederum nur unter bestimmten Annahmen notwendige Kriterien ableiten lassen. Die Fritz John-Bedingungen und die Karush-Kuhn-Tucker-Bedingungen benötigen als Voraussetzung die stetige Differenzierbarkeit der Zielfunktion sowie der Ungleichungs- und Gleichungsnebenbedingungen. Für hinreichende Kriterien, die aus diesen Bedingungen abgeleitet werden, müssen die Funktionen sogar zweimal stetig differenzierbar sein. Da dies in der Praxis nicht immer gegeben oder nicht nachprüfbar ist, lassen sich Verfahren, die auf diesen Bedingungen aufbauen, nicht auf allgemeine Probleme anwenden und werden daher nicht in der *EngineeringToolbox* benutzt. Eine ausführliche Betrachtung dieser Bedingungen findet sich in [1] und [11].

Es existieren jedoch noch weitere Ansätze zum Lösen einer EOA, z.B. die *Multikriterialisierung*, die zum ersten Mal in [17] beschrieben wurde. Bei diesem Verfahren werden einkriterielle Aufgaben durch Hinzunahme einer oder mehrerer geeigneter Zielfunktionen, so genannter *Helferfunktionen*, in multikriterielle Optimierungsaufgaben überführt, die dann mit geeigneten Verfahren der multikriteriellen Optimierung gelöst werden können. Ob solche Helferfunktionen für jede Aufgabe

existieren und wie man sie findet, ist nach [27] ein offenes Problem der Forschung. Im Folgenden sollen lediglich zwei mögliche Multikriterialisierungen aufgezeigt werden. Die erste Variante stammt aus [14]. Die Zielfunktion $f(x)$ einer einkriteriellen Minimierungsaufgabe wird darin als Summe mehrere Zielfunktionen $f_{r_1}(x)$ dargestellt, die dann in einer multikriteriellen Optimierung minimiert werden. Analog kann die Zielfunktion auch als Produkt mehrerer Zielfunktionen $f_{r_2}(x)$ dargestellt werden. Das Problem (2.2) wird also zu

$$\begin{aligned} f_{r_1}(x) \to min,\ r_1 = 1,2,...,m_1 & & f_{r_2}(x) \to min,\ r_2 = 1,2,...,m_2 \\ f(x) = \sum_{r_1=1}^{m_1} f_{r_1}(x) & \text{oder} & f(x) = \prod_{r_2=1}^{m_2} f_{r_2}(x) \\ x \in X \subseteq \mathbb{R}^n & & x \in X \subseteq \mathbb{R}^n \end{aligned}$$

Die Schwierigkeit liegt darin, eine geeignete Aufteilung der ursprünglichen Zielfunktion zu finden, so dass die multikriterielle Aufgabe leicht zu lösen ist. Es würde sich dabei beispielsweise anbieten, Teile der Zielfunktion, die nur von einer Variable abhängen, zu separieren.
Eine zweite Möglichkeit ist aus [20] entnommen und besteht darin, die restringierte EOA

$$\begin{aligned} f(x) &\to min \\ g_i(x) &\leq 0, \quad i = 1,2,...,k \\ x &\in \mathbb{R}^n \end{aligned}$$

in eine unrestringierte multikriterielle Aufgabe der Form

$$\begin{aligned} f(x) \to min & & f(x) \to min \\ \sum_{i=1}^{k} max\,(0, g_i(x)) \to min & \text{oder} & max\,(0, g_i(x)) \to min,\ i = 1,2,...,k \\ x \in \mathbb{R}^n & & x \in \mathbb{R}^n \end{aligned}$$

umzuwandeln. Die so erhaltene Aufgabe hat den Vorteil, dass sie keine Nebenbedingungen besitzt. Es ist jedoch darauf zu achten, dass die Zielfunktionen, die aus den Ungleichungsnebenbedingungen entstanden sind, 0 sein müssen, da sonst die gefundene Lösung unzulässig ist. Sie haben deshalb eine höhere Priorität gegenüber der eigentlichen Zielfunktion $f(x)$.

Weitere Verfahren für die Berechnung einer optimalen Lösung, die auch in der *EngineeringToolbox* eingesetzt werden, sind in Kapitel 3 beschrieben.

2.3 Multikriterielle Optimierungsaufgaben

Bei der *multikriteriellen* Optimierung, auch *Vektor-*, *Poly-* oder *Pareto-Optimierung* genannt, werden nicht nur eine, sondern mehrere Zielfunktionen betrachtet. Da sich die Ziele in der Regel widersprechen, z.B.

- soll bei einem PKW die Leistung maximiert und der Verbrauch minimiert werden
- oder eine Firma will die Qualität ihrer Produkte erhöhen, aber gleichzeitig die Produktionskosten senken,

kann man nicht mehr von *der* besten Lösung bzgl. einer OA sprechen, es müssen vielmehr Kompromisse zwischen den einzelnen Zielen eingegangen werden. Dazu werden Varianten betrachtet, die "besser" oder zumindest "nicht schlechter" als andere Varianten sind. Um dieses "besser" zu definieren, müssen zunächst einige Begriffe eingeführt werden, welche aus [6], [12], [22] und [30] entnommen wurden.

2.3.1 Begriffe und Definitionen

Definition 2.3: Multikriterielle Optimierungsaufgabe (MOA)
Eine *multikriterielle Optimierungsaufgabe* mit $m,\ m \geq 2$, Zielfunktionen $f_1, f_2, ..., f_m$ kann in Analogie zu (2.4) wie folgt geschrieben werden:

$$\begin{aligned} f_r(x) &\rightarrow min,\ \ r = 1, 2, ..., m \\ g_i(x) &\leq 0,\ \ i = 1, 2, ..., k \\ h_j(x) &= 0,\ \ j = 1, 2, ..., l \\ x &\in \mathbb{R}^n \end{aligned} \tag{2.6}$$

Fasst man die Zielfunktionen und Nebenbedingungen zu Vektoren zusammen erhält man die Kurzschreibweise von (2.6):

$$\begin{aligned} f(x) &\rightarrow min \\ g(x) &\leq 0 \\ h(x) &= 0 \\ x &\in \mathbb{R}^n \end{aligned} \tag{2.7}$$

mit $f(x) = (f_1(x), f_2(x), ..., f_m(x))^T$, $g(x) = (g_1(x), g_2(x), ..., g_k(x))^T$ und
$h(x) = (h_1(x), h_2(x), ..., h_l(x))^T$

Als nächstes wird eine Relation eingeführt, die es ermöglicht, zulässige Punkte miteinander zu vergleichen.

Definition 2.4: Dominanzrelation
Gegeben seien m zu minimierende Funktionen $f_r,\ r = 1, 2, ..., m$, und eine nichtleere Menge X. Ein Punkt $x \in X$ *dominiert* einen Punkt $y \in X$, kurz $x \prec y$, wenn gilt:

$$f_i(x) \leq f_i(y),\ \forall i \in \{1, 2, ..., m\}\ \wedge\ \exists j \in \{1, 2, ..., m\} :\ f_j(x) < f_j(y) \tag{2.8}$$

Bemerkung:
Die Dominanzrelation ist nicht reflexiv und nicht symmetrisch, aber asymmetrisch und transitiv. Sie ist also eine strikte Halbordnung.

Durch diese Relation lassen sich nun die Punkte $x \in X$ einer MOA charakterisieren.

Definition 2.5:
Existiert kein $y \in X$ mit $y \prec x$, so heißt der Punkt x *nicht-dominiert*, andernfalls nennt man ihn *(von y) dominiert*. Alle Punkte, die nicht-dominiert sind, nennt man auch *paretooptimal* oder *effizient* und können zur *(Edgeworth-)Pareto-* bzw. *Effizienzmenge* P zusammengefasst werden.

Auch im Wertebereich der Zielfunktionen lassen sich jetzt bestimmte Punkte charakterisieren. So wird die Menge $P_f = \{f(p) \mid p \in P\}$ der Funktionswerte der paretooptimalen Lösungen als *Pareto-Front* bezeichnet. Ein Beispiel für die Pareto-Front für zwei zu minimierende Funktionen ist in Abb. 2.1 dargestellt. Man sieht, dass die Front nicht notwendigerweise zusammenhängend sein muss, auch ist sie nicht zwingend konvex.
Minimiert man jede der m Zielfunktionen einzeln, so erhält man den *Idealpunkt* $f^*(x) = (f_1^*(x), f_2^*(x), ..., f_m^*(x))^T$ der OA. Einen Punkt $x^* \in X$, welcher alle Zielfunktionen minimiert, nennt man *perfekte* oder *ideale Lösung*, jedoch ist er in den meisten Fällen nicht zulässig oder nicht existent. Mit Hilfe des Idealpunktes lassen sich die unteren Grenzen der Pareto-Menge bestimmen, die oberen Grenzen entsprechen den Komponenten des *Nadirpunktes*, welcher jedoch für nichtlineare Probleme nicht analytisch ermittelt werden kann. Zur Illustration dient Abb. 2.1.

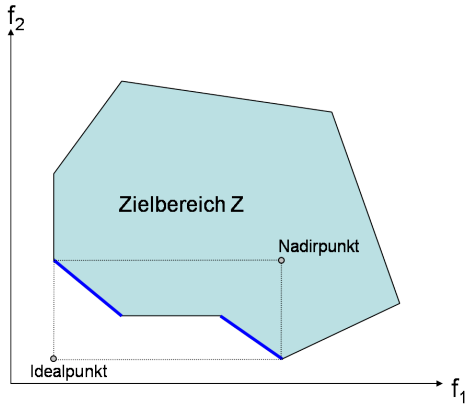


Abb. 2.1: Pareto-Front (blau) für zwei zu minimierende Funktionen mit Ideal- und Nadirpunkt

Das Ziel einer Optimierung besteht nun darin, die Pareto-Menge bzw. -Front zu bestimmen. Da dies in der Regel nicht exakt möglich ist, begnügt man sich meist damit, eine möglichst gute

Approximation dieser Mengen zu erhalten. Dabei können zwei verschiedene Teilziele betrachtet werden:

1. Die approximierte Menge soll möglichst nahe der Pareto-Front liegen.
2. Die Approximation soll gleichmäßig sein.

Die Veranschaulichung und die daraus resultierenden Probleme sind in Abb. 2.2 zu erkennen. Für das erste Ziel sind die gefunden Lösungen zwar gut, jedoch decken sie nur einen kleinen Teil der Pareto-Front ab. Bei einer gleichmäßigen Approximation hingegen sind die Lösungen gut verteilt, können aber unter Umständen weit von der eigentlichen Pareto-Front entfernt liegen.

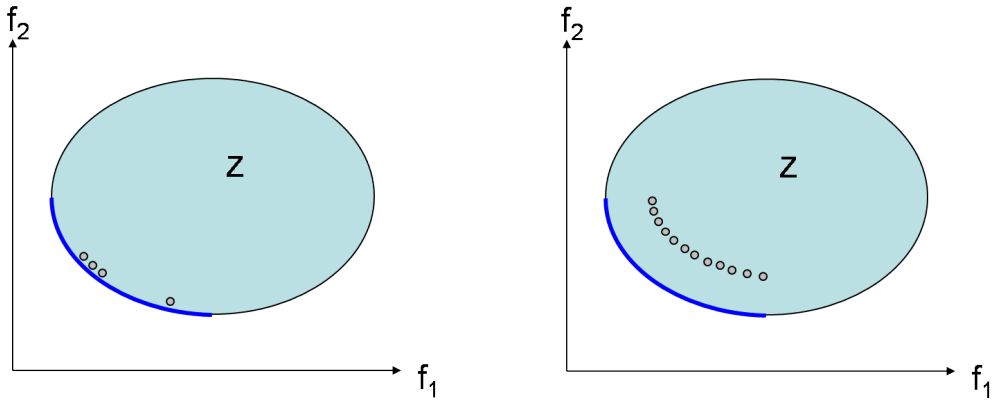


Abb. 2.2: Pareto-Front (blau) mit Approximation (graue Punkte) für Teilziel 1 (links) und Teilziel 2 (rechts)

Die verschiedenen Verfahren, die für MOA entwickelt wurden, werden in Kapitel 3 näher erläutert. Eine weitere Möglichkeit, Vektoroptimierungsaufgaben zu lösen, ist die Rückführung auf einkriterielle Probleme. Diese Transformationen führen zu Ersatzaufgaben der eigentlichen OA und werden auch als *klassische Lösungsmethoden* bezeichnet. Der nachfolgende Abschnitt beschäftigt sich mit zwei dieser Methoden, da diese auch in der *EngineeringToolbox* eingesetzt werden.

2.3.2 Klassische Lösungsmethoden

Es gibt viele verschiedene Transformationen zur Überführung multikriterieller Probleme in einkriterielle, im Nachfolgenden werden jedoch nur die zwei gebräuchlichsten näher betrachtet. Diese sind zum einen die *gewichtete Summe* (engl. *weighted sum*) und zum anderen die ϵ-*Constrained-Methode*. Beide Methoden haben Vor- und Nachteile, weshalb keine Aussage getroffen werden kann, welche der beiden Verfahren bevorzugt werden sollte. In beiden Fällen kann aber die transformierte Aufgabe mit Verfahren der einkriteriellen Optimierung gelöst werden und die Ergebnisse können auf die Ausgangsaufgabe übertragen werden.
Andere klassische Lösungsmethoden sind z.B. die Methode des gewichteten Abstandes, die Benson-

Methode, die Zielprogrammierung oder die Verwendung einer Nutzenfunktion. Sie werden u.a. in [6] und [22] näher beschrieben werden.

Gewichtete Summe

Bei der gewichteten Summe werden die m Zielfunktionen einer MOA mit einem geeigneten Gewichtsvektor $w = (w_1, w_2, ..., w_m)$ bewertet und zusammengefasst. Es entsteht eine Zielfunktion f_w der Form

$$f_w(x) = \sum_{r=1}^{m} w_r f_r(x),$$

die dann minimiert wird. Die Zielfunktionen sollten vor ihrer Wichtung jedoch normiert werden, da ihre Wertebereiche unter Umständen sehr unterschiedlich sein können und der Einfluss der Gewichtsvektoren dadurch verfälscht werden würde. Die Normierung ist allerdings nicht immer möglich, da dazu Kenntnisse über die Wertebereiche vorhanden sein oder aufwändige Testrechnungen durchgeführt werden müssen.
In der Literatur findet man für die Gewichte meist die Forderung, dass nur Werte zwischen 0 und 1 so verwendet werden, dass deren Summe 1 ergibt, d.h.:

$$\sum_{r=1}^{m} w_r = 1, \ w_r \in [0,1]$$

Es entsteht also folgendes Optimierungsproblem:

$$\begin{aligned} f_w(x) = & \sum_{r=1}^{m} w_r f_r(x) \rightarrow min \\ & g(x) \leq 0 \\ & h(x) = 0 \\ & \sum_{r=1}^{m} w_r = 1 \\ & x \in \mathbb{R}^n \\ & w_r \in [0,1], \ \forall r = 1, 2, ..., m \end{aligned} \tag{2.9}$$

Durch das Lösen dieser Aufgabe erhält man einen Punkt der Pareto-Menge. Es lässt sich zeigen, dass alle Lösungen von (2.9) für positive Gewichtsvektoren paretooptimal sind und für konvexe Aufgaben sogar alle Punkte der Pareto-Menge durch geeignete Wahl der Gewichtsvektoren berechnet werden können. Durch verschiedene Gewichtsvektoren können außerdem unterschiedliche Punkte erzeugt werden, wodurch die Pareto-Front besser approximiert wird (vgl. Abb. 2.3). Das Problem besteht allerdings darin, eine geeignete Wahl der Gewichtsvektoren zu finden. In den meisten Fällen werden die Gewichte zufällig z.B. mit Hilfe der *Latin-Hypercubes-Methode* (siehe [30]) bestimmt, es lässt sich jedoch auch eine eigene Wahl, die z.B. aus der Erfahrung oder den jeweiligen Interessen herrührt, treffen.

Die Methode der gewichteten Summe lässt sich leicht realisieren, der Nachteil besteht allerdings darin, dass konkave Abschnitte des Zielbereiches nicht erreicht werden können und dadurch in einem solchen Fall niemals die gesamte Pareto-Front ermittelt werden kann (vgl. Abb. 2.4, bei der der Bereich zwischen A und B, der zur Pareto-Front gehört, nicht erreicht werden kann).

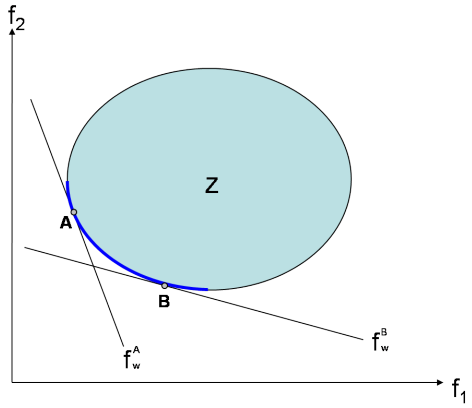


Abb. 2.3: Zwei gewichtete Zielfunktionen mit zwei unterschiedlichen Lösungen

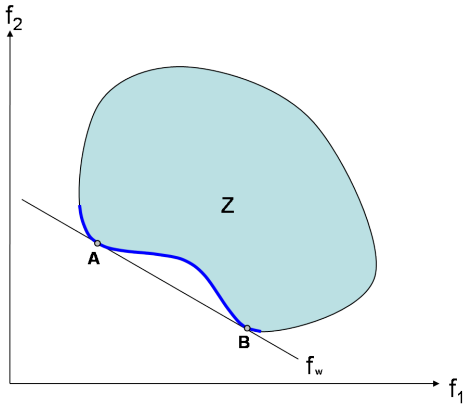


Abb. 2.4: Gewichtete Zielfunktion für einen konkaven Zielbereich

ϵ-Constrained-Methode

Bei der ϵ-Constrained-Methode wird eine der m Zielfunktionen der MOA ausgewählt, welche zu minimieren ist. Die restlichen $m-1$ Zielfunktionen gehen als Ungleichungsnebenbedingungen in die Optimierungsaufgabe ein:

$$\begin{aligned} f_s(x) &\rightarrow min,\ s \in \{1,2,...,m\} \\ f_r(x) &\leq \epsilon_r,\ r = 1,2,...,m,\ r \neq s \\ g(x) &\leq 0 \\ h(x) &= 0 \\ x &\in \mathbb{R}^n \end{aligned} \tag{2.10}$$

Dabei ist es wichtig, die oberen Schranken ϵ_r für jede Zielfunktion geeignet zu wählen. Ist ϵ_r zu klein, so wird die zulässige Menge leer und man erhält keine Lösung für das Optimierungsproblem. Wird für ϵ_r ein zu großer Wert angesetzt, liefert die Ungleichung keine Einschränkung an den zulässigen Bereich und könnte somit weggelassen werden. In Abb. 2.5 erkennt man, dass Schranke ϵ_2^A einen leeren zulässigen Bereich erzeugt und die Schranke ϵ_2^E keine Einschränkung bildet, wodurch die Schranken ϵ_2^D und ϵ_2^E den gleichen Punkt (D) liefern. In der schwierigen Wahl der Schranken liegt das Hauptproblem dieser Methode. Eine Möglichkeit besteht darin, die Zielfunktionen einzeln zu minimieren bzw. zu maximieren, wodurch der Bereich für die Schranken bestimmt werden kann. Dies bedeutet jedoch einen erheblichen Mehraufwand. Außerdem wird die Aufgabe im Gegensatz

zur Methode der gewichteten Summe vergrößert, da bei m Zielfunktionen $m-1$ Nebenbedingungen zur Ersatzaufgabe hinzukommen. Der Vorteil der ϵ-Constrained-Methode besteht darin, dass auch bei konkaven zulässigen Bereichen alle paretooptimalen Punkte erreicht werden können (siehe Punkt B und C in Abb. 2.5) und somit durch eine geeignete Wahl verschiedener Schranken und/oder Zielfunktionen die gesamte Pareto-Front approximiert werden kann.

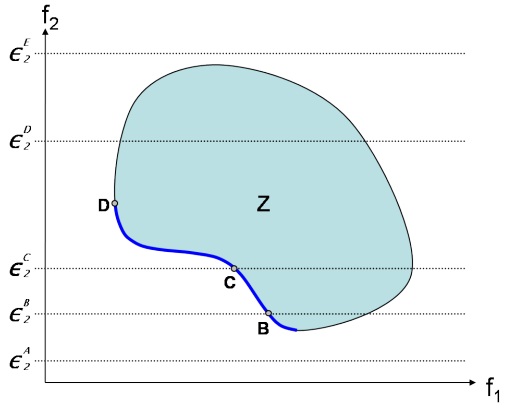


Abb. 2.5: Fünf verschiedene Schranken für die Zielfunktion f_2

2.4 Diskrete Optimierungsaufgaben

Da bei Optimierungsaufgaben aus der Praxis die Parameter nicht selten für Stückzahlen, Arbeitskräfte oder Entscheidungen (z.B. ja/nein) stehen, spielt die diskrete Optimierung eine bedeutende Rolle in der Anwendung der Optimierungstheorie. Zu den diskreten Optimierungsaufgaben (DOA) gehören u.a. Zuordnungsprobleme, Produktionsplanungen, das Rucksack- und das Rundreiseproblem. DOA können außerdem nach der Art ihrer Parameter eingeteilt werden. Sind z.B. alle Parameter ganzzahlig, so spricht man von einer *rein-ganzzahligen Optimierung*. Sind einige der Variablen stetig, wird die Aufgabe als *gemischt-ganzzahlig* bezeichnet. Sind die Parameter zwar diskret, aber nicht ganzzahlig, so sprich man allgemeiner von einer *rein-* bzw. *gemischt-diskreten Aufgabe*. Eine besondere Rolle spielen die *kombinatorischen Optimierungsaufgaben* (KOA), bei denen die zulässige Menge nur aus endlich vielen Punkten besteht. Einen Spezialfall der kombinatorischen Optimierung bilden die *Booleschen* bzw. *0-1-Optimierungsaufgaben*, bei denen die Parameter nur die Werte 0 oder 1 annehmen können.
Durch die Diskretheitsforderung an alle oder einige Parameter ist die zulässige Menge nicht zusammenhängend und nicht konvex, wodurch die Verfahren der stetigen Optimierung nicht unmittelbar angewandt werden können. Es müssen daher neue Lösungsideen entwickelt werden, welche im Folgenden kurz beschrieben werden sollen.

Eine Möglichkeit zur Lösung kombinatorischer Aufgaben besteht darin, eine *vollständige Enumeration* durchzuführen, d.h. für alle zulässigen Punkte wird der Zielfunktionswert berechnet und anschließend der beste dieser Werte ermittelt. Jedoch bedeutet dies schon für wenige Parameter einen erheblichen Rechenaufwand (vgl. Bsp. 2.1), wodurch dieses Verfahren praktisch nicht anwendbar ist.

Beispiel 2.1:
Gegeben ist eine DOA mit 10 Parametern, die jeweils einen Wert aus der Menge $\{1, 2, ..., 15\}$ annehmen können. Es entstehen bei einer vollständigen Enumeration folglich 10^{15} zu berechnende Varianten. Ein Rechner, der 1 Mio. Funktionsauswertungen pro Sekunde ausführen kann, benötigt dafür 10^9 Sekunden $\approx$ 32 Jahre, eine inakzeptable Rechenzeit.

Einen anderen Ansatz zur Lösung diskreter Optimierungsaufgaben erhält man durch die Transformation dieser in eine kontinuierliche Aufgabe, indem man die Diskretheitsbedingungen unbeachtet lässt. Löst man die so entstandene Aufgabe mit Verfahren der stetigen Optimierung, kann die erhaltene Lösung anschließend auf Werte aus dem zulässigen Bereich gerundet werden. Das Problem, welches sich daraus ergibt, liegt in der Art des Rundens. So kann zunächst nicht gesagt werden, ob auf- oder abzurunden ist und ob dann die gerundete Lösung wirklich ein Optimum oder überhaupt zulässig ist. Das nachfolgende Bsp. 2.2 aus [18] verdeutlicht dieses Problem.

Beispiel 2.2:
Es ist die folgende rein-ganzzahlige OA zu lösen:

$$\begin{aligned} f(x) = &\quad x_1 - 3x_2 + 3x_3 &&\rightarrow max \\ g_1(x) = &\quad 2x_1 + x_2 - x_3 &&\leq 4 \\ g_2(x) = &\quad 4x_1 - 3x_2 &&\leq 2 \\ g_3(x) = &\quad -3x_1 + 2x_2 + x_3 &&\leq 3 \\ &\quad x_i \in \mathbb{N},\ i = 1, 2, 3 && \end{aligned} \tag{2.11}$$

Löst man diese Aufgabe ohne die Diskretheitsbedingung, so erhält man die optimale Lösung $x^* = (x_1^*, x_2^*, x_3^*) = (0.5, 0, 4.5)$. Egal wie man diese Lösung rundet, für (2.11) ist sie unzulässig.
Die optimale Lösung der diskreten Aufgabe ist $x_D^* = (x_{D_1}^*, x_{D_2}^*, x_{D_3}^*) = (2, 2, 5)$.

Eine weitere Idee zur Lösung ganzzahliger KOA, welche leicht für reellwertige KOA erweitert werden kann, stammt aus [1]. Dazu wird die KOA zunächst in eine Boolesche Aufgabe überführt: Angenommen die Variable x_i nimmt nur die I Werte $\alpha_i^{(1)}, \alpha_i^{(2)}, ..., \alpha_i^{(I)}$ an, dann kann x_i durch

$$x_i = \alpha_i^{(1)} * x_i^{(1)} + \alpha_i^{(2)} * x_i^{(2)} + ... + \alpha_i^{(I)} * x_i^{(I)}$$

mit $x_i^{(1)} + x_i^{(2)} + ... + x_i^{(I)} = 1$ und $x_i^{(1)}, x_i^{(2)}, ..., x_i^{(I)} \in \{0, 1\}$ ersetzt werden. Führt man diese Transformation für alle Parameter durch, so erhält man eine 0-1-Optimierungsaufgabe. Für diese kann die Boolesche Variable x_j durch eine stetige Variable x_J ersetzt werden, wenn man die Gleichung $x_J = x_J^2$ als Nebenbedingung zur Aufgabe hinzufügt. Somit lässt sich jede KOA in eine

nichtlineare kontinuierliche Aufgabe umformen, ohne dass die Lösungsmenge verändert wird. Es ist dabei jedoch zu beachten, dass sich pro ersetzter Variable die Anzahl der Parameter in der ersten Transformation um $I-1$ erhöht und im ersten und zweiten Schritt der Umformung jeweils eine Gleichungsnebenbedingung zur ursprünglichen Aufgabe hinzukommt, wodurch sich der Rechen- und Speicheraufwand gegenüber der Ausgangsaufgabe extrem erhöht. Außerdem ist dieses Verfahren nur dann sinnvoll, wenn ein effektiver Algorithmus zur Lösung nichtlinearer Optimierungsaufgaben vorhanden ist.

Da sich die oben vorgestellten Ideen praktisch meist nicht einsetzen lassen, wurden weitere Verfahren zur Lösung diskreter Optimierungsaufgaben entwickelt, die sich in *exakte* und *heuristische Verfahren* unterteilen lassen. Die folgenden Ausführungen zu diesen Lösungsmethoden sind an [2], [7], [18] und [26] angelehnt.

2.4.1 Exakte Verfahren

Exakte Verfahren sind theoretisch genaue Verfahren, für die bewiesen werden kann, dass sie die OA in endlich vielen Schritten exakt lösen oder ihre Unlösbarkeit erkennen. Zu ihnen zählen u.a. das *Schnittprinzip*, die *Branch-and-Bound-Methode*, der additive Algorithmus von Balas und die Methode von Faure und Malgrange. Das Problem vieler exakter Verfahren besteht darin, dass möglicherweise die Diskretheit und damit die Zulässigkeit der Variablen nicht erkannt werden kann, da sich aufgrund der Rechnerungenauigkeit keine exakten Werte bestimmen lassen. Auch der meist hohe Rechenaufwand und die häufige Unkenntnis genauer Eingangsdaten der Aufgabe rechtfertigen den Einsatz exakter Verfahren nur selten. Nichtsdestotrotz sollen zwei dieser Methoden kurz beschrieben werden, da aus ihnen z.T. heuristische Verfahren abgeleitet und sie zur Einschätzung der Lösungen aus Näherungsverfahren genutzt werden können. Für weitere Ausführungen zu den exakten Lösungsverfahren empfiehlt sich [18].

Schnittprinzip

Die Grundlage des Schnittprinzips bildet die *Relaxation* der diskreten Ausgangsaufgabe. Dazu muss dieser Begriff zunächst definiert werden.

Definition 2.6: Relaxation
Gegeben sei die Optimierungsaufgabe (2.2). Die Aufgabe

$$\begin{aligned} &\widehat{f}(x) \rightarrow min \\ &x \in \widehat{X} \subseteq \mathbb{R}^n \end{aligned} \tag{2.12}$$

heißt *Relaxation* von (2.2), wenn $X \subseteq \widehat{X}$ und $\widehat{f}(x) \leq f(x),\ \forall x \in X$, gilt.

Mit Hilfe der Relaxation können nun Aussagen über den Optimalwert der DOA getroffen werden.

Satz 2.1:
Gegeben sind die DOA (2.2) und eine zugehörige Relaxation (2.12), welche beide lösbar sind, d.h. $f^* = \min\limits_{x \in X} f(x)$ und $\widehat{f}^* = \min\limits_{x \in \widehat{X}} \widehat{f}(x)$ existieren. Dann gilt:

1. $\widehat{f}^* \leq f^*$
2. $\widehat{f}(x^\circ) = \widehat{f}^* \wedge x^\circ \in X \wedge \widehat{f}(x^\circ) = f(x^\circ) \Longrightarrow f(x^\circ) = f^*$

Interpretation:

1. $\widehat{f}^*$ ist eine untere Schranke für die optimale Lösung der Ausgangsaufgabe (2.2).
2. entspricht einem Optimalitätskriterium: Ist die optimale Lösung x° von (2.12) zulässig für (2.2) und sind die Zielfunktionswerte in diesem Punkt gleich, so ist x° auch die optimale Lösung der DOA.

Beweis:

1. Wie man leicht nachvollziehen kann, gelten die folgenden Beziehungen:

$$\widehat{f}^* = \min_{x \in \widehat{X}} \widehat{f}(x) \leq \min_{x \in X} \widehat{f}(x) \leq \min_{x \in X} f(x) = f^*$$

2. indirekter Beweis:
   Angenommen es gilt $f(x^\circ) \neq f^*$, dann ist entweder (a) $f(x^\circ) > f^*$ oder (b) $f(x^\circ) < f^*$.
   $f(x^\circ) = \widehat{f}(x^\circ) = \widehat{f}^* \leq f^*$ führt zu einem Widerspruch zu (a).
   (b) $f(x^\circ) < f^* = \min\limits_{x \in X} f(x)$ führt ebenfalls zu einem Widerspruch, da mit $x^\circ \in X$ f^* kein Minimum in X wäre, was jedoch vorausgesetzt wurde.
   Damit ist die Annahme falsch und es folgt die Behauptung $f(x^\circ) = f^*$.

□

Die Idee der Schnittmethoden besteht nun darin, ausgehend von der Relaxation der zu lösenden Aufgabe, eine Folge von k, $k \geq 1$, Aufgaben (P_k) zu bilden, deren zulässiger Bereich sukzessive mittels Schnittbedingungen verkleinert wird, bis die optimale Lösung der Ausgangsaufgabe gefunden wurde. Die Schnitte werden dabei immer ausgehend von der vorangegangenen Aufgabe gebildet. Außerdem sollten die Optimierungsaufgaben (P_k) so konstruiert werden, dass sie leicht zu lösen sind, eine optimale Lösung besitzen und sich in der nachfolgenden Form darstellen lassen:

$$\begin{aligned} \widehat{f}(x) &\rightarrow min \\ x &\in X_k \end{aligned} \tag{2.13}$$

mit $X_{k+1} = X_k \backslash S(x^k)$ und x^k optimale Lösung von (P_k), sowie (P_1) ist eine Relaxation zur Ausgangsaufgabe.

Die Schnitte $S(x^k)$ werden so gebildet, dass $x^k \in S(x^k)$ und $S(x^k) \subset X_k$ gilt, so dass der zulässige Bereich der nächsten Aufgabe P_{k+1} echt kleiner, aber nicht leer ist. Des Weiteren muss noch geregelt werden, welche Elemente (außer x^k) aus X_k ausgeschlossen bzw. "abgeschnitten" werden dürfen. Prinzipiell dürfen aus $X_k \backslash X$ beliebige Elemente entfernt werden, nur für zulässige Elemente der Ausgangsaufgabe gibt es Einschränkungen. Ist x^k unzulässig für die Ausgangsaufgabe, d.h. $x^k \notin X$, so dürfen keine Elemente aus X entfernt werden, es muss also $S(x^k) \cap X = \emptyset$ gelten. Ist aber $x^k \in X$, so können all die zulässigen Lösungen abgeschnitten werden, deren Zielfunktionswert größer oder gleich $f(x^k)$ ist. Es muss sich dabei aber x^k als mögliche optimale Lösung für die DOA gemerkt werden.
Die beste der optimalen Lösungen der Teilaufgaben (P_k), die auch zulässig für die Ausgangsaufgabe ist, ist dann die optimale Lösung der DOA. Für den Spezialfall $f(x) = \widehat{f}(x)$ lässt sich sogar zeigen, dass die erste dieser Lösungen, die zulässig ist, auch die optimale Lösung der DOA ist.
Um aus dem Schnittprinzip ein programmierbares Verfahren zu erhalten, müssen die Relaxationen und die Schnitte genauer beschrieben werden. Solche Beschreibungen wurden u.a. von Gomory, dem Begründer des Schnittprinzips, Dantzig, Dalton und Llewellyn aufgestellt und können in [18] nachgelesen werden.
Durch den einfachen Lösungsgedanken des Schnittprinzips lassen sich davon abgeleitete Algorithmen leicht programmieren. Sie eignen sich besonders zur Lösung rein- oder gemischt-ganzzahliger Aufgaben, jedoch weniger für KOA. Das Problem der Schnittprinzipien liegt in der Effektivität der daraus abgeleiteten Verfahren. So kann die Endlichkeit der Algorithmen nur unter bestimmten Voraussetzungen nachgewiesen werden und es gibt keine brauchbaren Beschränkungen der Anzahl der durchzuführenden Schnitte. Auch der wachsende Speicherplatzbedarf mit steigender Schnittzahl ist problematisch, da mit jedem Schnitt eine Nebenbedingung zur Ausgangsaufgabe hinzukommt.

Branch-and-Bound-Methode

Die Branch-and-Bound-Methode (BBM) gehört zu den Entscheidungsbaum-Methoden, bei der der zulässige Bereich, die Wurzel des Baumes, in Teilgebiete aufgeteilt wird, welche die Knoten dieses Baumes bilden. Die Menge der zulässigen Punkte wird also sukzessive in disjunkte Teilmengen zerlegt, die anschließend weiter verzweigt (engl. *branch*) werden. Mittels geeigneter unterer und oberer Schranken (engl. *bounds*) können dann einige dieser Zweige aus der zu untersuchenden Lösungsmenge ausgeschlossen werden; die BBM ist somit eine *unvollständige Enumeration*. Für KOA ist die Endlichkeit dieser Methode bewiesen, da der zulässige Bereich nach endlich vielen Schritten in einelementige Teilmengen zerlegt ist, die nicht weiter verzweigt werden müssen.
Auch die BBM ist nur ein Lösungsprinzip, so dass zur Erweiterung zu einem programmierbaren Algorithmus für eine konkrete DOA die Verzweigungsregeln und Schrankenberechnungen vorher definiert werden müssen. Die ersten Ausarbeitungen stammen von Land und Doig, später beschäftigten sich auch Little, Murty, Sweeney und Karel mit der BBM und lieferten konkrete Berech-

nungsvorschriften für die Verzweigungen und Schranken bei Rundreiseproblemen. Die Algorithmen von Land und Doig sowie zum Rundreiseproblem sind u.a. in [18] zu finden.
Die Verfahren, die auf der BBM beruhen, sind im Vergleich zu den Schnittprinzipien nicht so anfällig für Rundungsfehler und ihr Verlauf ist besser vorherzusagen, wodurch sie für Aufgaben mit einer nicht zu großen Anzahl an Variablen zu bevorzugen sind. Für hochdimensionale Probleme können die Rechenzeit und der Speicherplatzbedarf jedoch erheblich groß werden, da die Effektivität der Algorithmen entscheidend von der Art der Verzweigung und den Schranken, die möglichst früh ganze Zweige ausschließen sollten, abhängt. Ein weiteres Problem liegt darin, dass die optimale Lösung zwar schnell gefunden, jedoch ihre Optimalität nicht sofort nachgewiesen werden kann. Ein vorzeitiger Abbruch des Algorithmus liefert aber im Gegensatz zu den Schnittverfahren eine zulässige Näherungslösung. Besonders eignen sich die BBM für KOA, speziell für Boolesche Aufgaben.

Eine neuere Methode zur Lösung von DOA ist eine Kombination aus Schnittmethoden und BBM, die so genannten *Branch-and-Cut-Methoden* (BCM). Zunächst wird auf die Ausgangsaufgabe ein Schnittverfahren angewandt, d.h. es werden ein oder mehrere Schnitte zu der Aufgabe hinzugefügt, um die dann entstandene Aufgabe mit einem Verfahren der BBM zu lösen. Es können auch während der Verzweigungen weitere Schnitte eingefügt werden, die entweder nur für einen Knoten oder aber für alle Teilaufgaben gültig sind. Auch bei dieser Methode gibt es viel Spielraum zur Konkretisierung der Schnitte, Schranken, Verzweigungen und ob ein Schnitt oder eine Verzweigung durchgeführt werden soll.
Die BCM sind zur Zeit noch in der Entwicklungsphase, es lässt sich aber schon jetzt sagen, dass Verfahren nach diesem Prinzip effektiver arbeiten können als Verfahren, die nur auf dem Schnittprinzip oder der BBM beruhen.

2.4.2 Heuristische Verfahren

Durch den meist exponentiellen Rechenaufwand und den hohen Speicherplatzbedarf exakter Verfahren werden diese bei großdimensionierten Optimierungsaufgaben selten eingesetzt. Auch lassen sich die Aufgaben manchmal von vornherein nicht völlig exakt beschreiben und einige der aufgestellten Eingangsgrößen und Forderungen beruhen nur auf Schätzungen oder sind Näherungen. In manchen Fällen wird die Aufgabe auch aus praktischen Gründen nur vereinfacht dargestellt. Daher lohnt sich der Einsatz solcher aufwändigen Verfahren zum Finden einer exakten Lösung oft nicht und es werden heuristische Verfahren, auch *Näherungs-* oder *Approximationsverfahren* genannt, eingesetzt. Sie haben einen polynomialen Rechenaufwand und einen kleinen Speicherplatzbedarf, jedoch finden sie in der Regel nur suboptimale Lösungen, d.h. Lösungen, die zulässig sind und deren Zielfunktionswert sich nur wenig vom Optimalwert unterscheidet. Meist lassen sich aber keine brauchbaren Schranken für die Abweichung der gefundenen Lösung zum Optimum angegeben und sollte ein heuristisches Verfahren eine optimale Lösung erreichen, so lässt sich dies in der Regel

nicht nachweisen.

Die verschiedenen Näherungsverfahren lassen sich in *Eröffnungsverfahren*, *Verbesserungsverfahren* und *Gesamtverfahren* unterteilen. Zu den bekanntesten Eröffnungsverfahren gehört der Greedy-Algorithmus, für den es unter gewissen Voraussetzungen recht gute Abschätzungen für die Qualität der gefundenen Lösung gibt. Die Eröffnungsverfahren dienen der Bestimmung einer ersten zulässigen Lösung, die dann mit Hilfe eines Verbesserungsvefahrens verbessert werden kann. Bei den Verbesserungsverfahren unterscheidet man zwischen *reinen Verbesserungsverfahren*, für die in jedem Schritt eine bessere Lösung gefunden werden muss, und *lokalen Suchverfahren* bzw. *stochastischen Näherungsverfahren*, die eine kurzzeitige Verschlechterung der Lösung zulassen. Das Problem reiner Verbesserungsverfahren liegt darin, dass sie nur lokale Optima finden und in diesen verbleiben. Durch das Zulassen einer kurzzeitigen Verschlechterung des Zielfunktionswertes hingegen kann ein lokales Optimum wieder verlassen werden, um weitere, bessere Lösungen zu suchen. Die Suche nach besseren Lösungen wird dabei auf eine Nachbarschaft der zur Zeit besten Lösung beschränkt, daher nennt man diese Verfahren auch *Nachbarschafts-Suchverfahren*. Sowohl für die Bestimmung der Nachbarschaft als auch für die Auswahl der nächsten, besseren Lösung gibt es unterschiedliche Strategien, wodurch sich wiederum eine Vielzahl verschiedener Verfahren ableiten lässt. Bespiele für lokale Suchverfahren sind Simulated Annealing, Tabu Search und der Sintflutalgorithmus.

Gesamtverfahren kombinieren das Finden einer zulässigen Startlösung und das Verbessern dieses Punktes. Zu ihnen gehören u.a. vorzeitig abgebrochene exakte Verfahren und naturanaloge Verfahren wie der Ameisenalgorithmus sowie evolutionäre und genetische Verfahren.

3 Optimierungsverfahren

In diesem Kapitel werden die in der *EngineeringToolbox* eingesetzten Optimierungsverfahren, welche in Kapitel 4 diskretisiert werden, näher beschrieben. Zunächst folgt ein kurzer Überblick über die verschiedenen Optimierungsverfahren, danach wird auf die betrachteten Verfahren ausführlicher eingegangen.

3.1 Überblick

Der nun folgende Überblick über die Einordnung der verschiedenen numerischen Verfahren der Optimierung wurde zu weiten Teilen aus [29], [30], [31] und [32] entnommen.
Optimierungsverfahren können zunächst in *deterministische* und *stochastische Verfahren*, auch *Heuristiken* genannt, eingeteilt werden. Die deterministischen Verfahren zeichnen sich dadurch aus, dass im gesamten Verlauf der Optimierung nie der Zufall eine Rolle spielt und stets bestimmte algorithmische Schritte in der gleichen Reihenfolge sowie Art und Weise angewandt werden. Es können also in jedem Schritt die Zwischenergebnisse vorhergesagt werden und man erhält auch bei mehrmaliger Anwendung eines solchen Verfahrens auf eine Aufgabe mit denselben Startwerten stets die gleiche Lösung. Dadurch lassen sich leichter Aussagen zur Konvergenz und zur Genauigkeit deterministischer Algorithmen treffen. Es besteht dadurch aber nicht die Chance, bei einem zweiten Durchlauf eine bessere Lösung zu erhalten, falls der Algorithmus im ersten Versuch keine optimale Lösung gefunden hat. Die einzige Möglichkeit, mehrere Lösungen zu ermitteln und somit mögliche lokale Optima auszuschließen, besteht darin, das Verfahren mit verschiedenen Startlösungen zu beginnen. Ist der Zielbereich der OA unbekannt, ist dies auch ein probates Mittel, um überhaupt eine brauchbare Lösung zu finden, da das Verhalten deterministischer Verfahren stark von der Wahl der Startlösung abhängt. Deterministische Algorithmen konvergieren in der Regel recht schnell, setzen sich aber häufig in einem lokalen Optimum fest und werden meist nur für spezielle Aufgabenklassen entwickelt, wodurch sie nicht allgemein einsetzbar sind. Zu den deterministischen Verfahren gehören u.a. die *Gradientenverfahren*, wie z.B. die Methode der *sequentiellen quadratischen Programmierung* (SQP) oder das *Newton-Verfahren*. Diese Methoden benötigen den Gradienten der Zielfunktion, welcher jedoch nicht für jede Aufgabe gegeben bzw. ermittelbar ist. Weitere deterministische Verfahren sind das *Dividing-Rectangles-* (DiRect) und das *Nelder-Mead-Verfahren*, welche in Abschnitt 3.2 näher beschrieben werden.
Zu den stochastischen Verfahren gehören alle zufallsbasierten Strategien. Der große Vorteil dieser Methoden besteht darin, dass ungünstige Bereiche des Suchraums zufällig verlassen werden können, um dann bessere Bereiche zu untersuchen. Dies gelingt jedoch meist nur auf Kosten der Konvergenzgeschwindigkeit. Ein weiterer Vorteil stochastischer Verfahren besteht darin, dass sie

nur wenige Informationen über die zu lösende OA benötigen. Sie eignen sich daher gut zur Lösung so genannter *Black-Box-Probleme*[1]. Das einfachste und bekannteste stochastische Verfahren ist die *Monte-Carlo-Methode*, bei der zufällig gleichverteilte Punkte aus dem zulässigen Bereich ausgewählt werden, deren Zielfunktionswert berechnet und der beste bestimmt wird. Zwei weitere große Gruppen der Heuristiken bilden die *Ersatzmodell-gestützten* (siehe 3.2.4) und die *naturanalogen Verfahren*, die natürliche Prozesse nachbilden und sich wiederum in *schwarmintelligente* und *evolutionäre Algorithmen* unterteilen lassen. Zu den schwarmintelligenten Verfahren gehören u.a. der *Ameisenalgorithmus* und die *Partikelschwarmoptimierung* (PSO), welche in 3.2.3 näher beschrieben wird. Beispiele für evolutionäre Algorithmen sind der gewöhnliche genetische Algorithmus (engl. *Ordinary Genetic Algorithm* (OGA)) mit seinen Erweiterungen MOGA (*Multipler* OGA) und PGA (*Probabilistischer* GA).
Eine Übersicht über ausgewählte Optimierungsverfahren und deren Einordnung liefert die Abbildung 3.1.

3.2 Spezielle Optimierungsverfahren

In diesem Abschnitt werden nun die Verfahren, die in Kapitel 4 diskretisiert werden, näher erläutert. Dabei wird auf den allgemeinen Ablauf des Verfahrens, die Anwendungsmöglichkeiten sowie Vor- und Nachteile dieser Methoden eingegangen. Das Dividing-Rectangles-Verfahren wird dabei ausführlicher beschrieben, da es neu in die *EngineeringToolbox* implementiert wurde.

3.2.1 Dividing-Rectangles-Verfahren

Das Dividing-Rectangles-Verfahren wurde aus der Lipschitz-Optimierung entwickelt und erstmals 1993 von Jones, Perttunen und Stuckman in [16] beschrieben. Weitere Betrachtungen sind in [3], [8], [9] und [19] zu finden, welche auch die Grundlage für die folgenden Ausführungen bilden.
Um das DiRect-Verfahren besser nachvollziehen zu können, wird zunächst die Lipschitz-Optimierung kurz erläutert. Mit dieser Methode wird das globale Minimum einer eindimensionalen Lipschitz-stetigen Funktion gefunden. Die Lipschitz-Stetigkeit ist in [19] wie folgt definiert:

Definition 3.1: Lipschitz-Stetigkeit
Die Funktion $f : [l, u] \to \mathbb{R}$ heißt *Lipschitz-stetig*, wenn eine Konstante $K \in \mathbb{R}^+$ existiert, so dass

$$\forall x, x^\circ \in [l, u] : |f(x) - f(x^\circ)| \leq K|x - x^\circ| \tag{3.1}$$

K wird daher auch als *Lipschitz-Konstante* bezeichnet.

[1] Black-Box-Probleme sind OA, bei denen die Zielfunktion nicht analytisch beschrieben werden kann. Es stehen insbesondere keine Informationen über Definitions- und Wertebereich der Zielfunktion zur Verfügung. Black-Box-Aufgaben treten häufig bei technischen Problemen auf (siehe [30]).

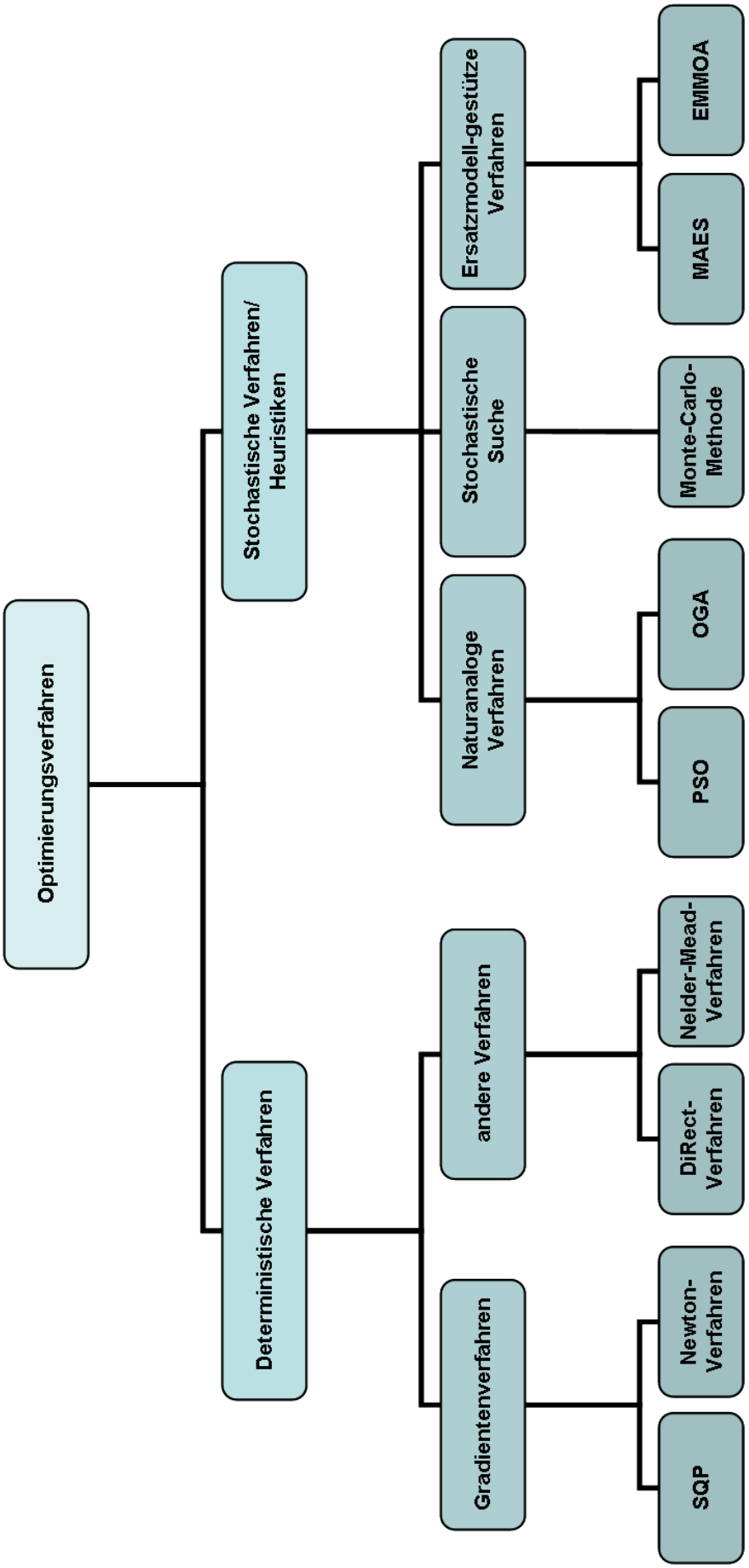


Abb. 3.1: Überblick über ausgewählte Optimierungsverfahren

Durch diese Definition können für alle Funktionswerte einer Lipschitz-stetigen Funktion $f(x)$ mit $x \in [a,b] \subseteq [l,u]$ zwei untere Schranken angegeben werden:

$$\begin{aligned} f(x) &\geq f(a) - K(x-a), \ \forall x \in [a,b] \\ f(x) &\geq f(b) + K(x-b), \ \forall x \in [a,b] \end{aligned} \tag{3.2}$$

Fasst man diese beiden Schranken zusammen, erhält man:

$$f(x) = max\left(f(a) - K(x-a), f(b) + K(x-b)\right) \tag{3.3}$$

Durch (3.3) entsteht eine V-förmige Beschränkung der Lipschitz-stetigen Funktion, wie sie in Abb. 3.2 zu sehen ist.

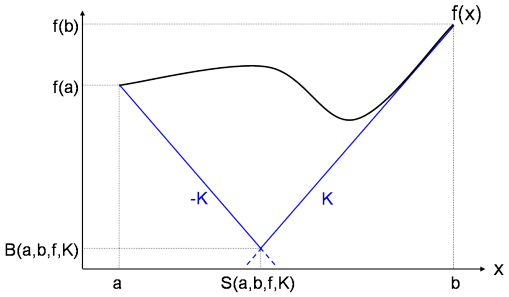


Abb. 3.2: Untere Schranke (blau) für eine Lipschitz-stetige Funktion

Der kleinste Wert B dieser Schranke ist die Spitze der V-förmigen Beschränkung und hängt, wie auch die Position S der Spitze, von den Rändern a und b sowie von der Funktion f und der Lipschitz-Konstante K ab. Um S zu berechnen, werden die beiden Schranken aus (3.2) gleichgesetzt und umgeformt:

$$\begin{aligned} f(a) - K(S-a) &= f(b) + K(S-b) \\ f(a) - f(b) &= K(S-b+S-a) = 2KS - K(a+b) \\ f(a) - f(b) + K(a+b) &= 2KS \end{aligned}$$

Damit ergibt sich für die Position der Spitze:

$$S(a,b,f,K) = \frac{a+b}{2} + \frac{f(a)-f(b)}{2K} \tag{3.4}$$

Den zugehörigen Wert B der Schranke erhält man durch Einsetzen von (3.4) in (3.3):

$$B(a,b,f,K) = \frac{f(a)+f(b)}{2} - \frac{K(b-a)}{2} \tag{3.5}$$

Aus den beiden Gleichungen (3.4) und (3.5) entwickelte Shubert den Algorithmus 3.1 zur Bestimmung des globalen Minimums einer Lipschitz-stetigen Zielfunktion mit Box-Restriktionen.

Algorithmus 3.1: Shubert-Algorithmus

Input: EOA mit der Box-Restriktion $l \leq x \leq u$, Lipschitz-Konstante K der Zielfunktion f

Output: globales Minimum der Aufgabe

1: Berechne die Funktionswerte an den beiden Intervallgrenzen l und u
2: Bestimme $x_1 = S(l, u, f, K)$ und teile das Ausgangsintervall $[l, u]$ in die beiden Teilintervalle $[l, x_1]$ und $[x_1, u]$
3: **repeat**
4: Berechne B für alle Teilintervalle
5: Wähle das Intervall mit dem kleinsten B und teile dieses Intervall an S
6: **until** Abbruchbedingung erfüllt

Die ersten drei Iterationen eines Shubert-Algorithmus sind in Abb. 3.3 beispielhaft dargestellt. Dabei hätte im zweiten Schritt auch das rechte Intervall geteilt werden können, da in diesem Fall die Werte für B gleich groß sind.

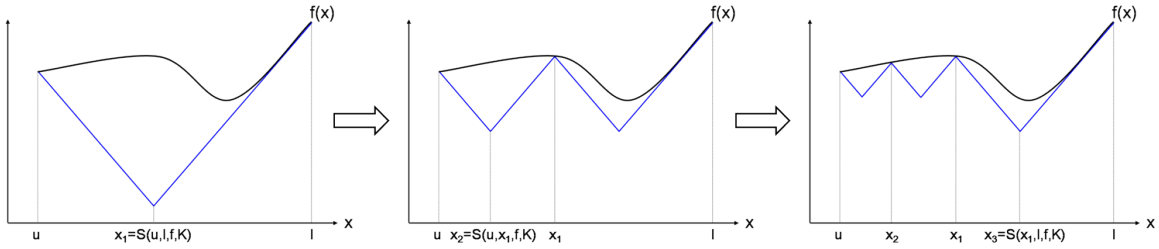


Abb. 3.3: Verlauf des Shubert-Algorithmus

Man kann erkennen, dass die V-förmigen Beschränkungen der einzelnen Intervalle die Funktion f von unten approximieren und die Approximation umso besser ist, je mehr Teilintervalle gebildet werden. Der Algorithmus wird daher abgebrochen, wenn eine gewünschte Genauigkeit tol der Approximation erreicht ist, d.h. für das aktuelle Minimum S gilt $|B - f(S)| < tol$, wobei B der zu S gehörige kleinste Wert der Schranke ist.

In Zeile 5 des Algorithmus 3.1 wird jeweils das kleinste B aller Intervalle gesucht. Dabei ist B genau dann klein, wenn mindestens einer der Summanden aus (3.5) klein ist. Für den ersten Summanden $\frac{f(a)+f(b)}{2}$ bedeutet dies, dass die Funktionswerte klein sind, d.h. im Bereich des Minimums liegen. Wird also ein Intervall mit dieser Eigenschaft gewählt, so wird lokal nach dem Optimum gesucht. Wird hingegen ein B gewählt, bei dem der zweite Summand $-\frac{K(b-a)}{2}$ klein ist, muss die Differenz zwischen a und b groß sein, es wird demnach global in einem großen Intervall gesucht. Die Konstante K stellt dabei einen Gewichtsfaktor zwischen lokaler und globaler Suche dar. Ist K groß, so wird mehr Wert auf eine globale Suche gelegt, für kleine K ist die Suche eher lokal. Daraus ergibt sich jedoch ein Problem des Shubert-Algorithmus: Da die Lipschitz-Konstante meist nur sehr grob abgeschätzt werden kann, d.h. relativ groß ist, konzentriert sich der Algorithmus auf eine globale Suche, was zu einer langsamen Konvergenz führt. Ein weiteres Problem liegt im ersten Schritt des

Algorithmus, bei dem die Funktionswerte an den Intervallgrenzen berechnet werden. Für Funktionen mit mehreren Variablen steigt der Aufwand der Berechnung exponentiell, da für n Variablen 2^n Werte berechnet werden müssen.

Diese Nachteile des Shubert-Algorithmus motivierten nach Finkel (vgl. [8]) die Entwicklung des DiRect-Verfahrens. Im Gegensatz zur Methode nach Shubert werden nicht die Randpunkte, sondern die Mittelpunkte der Intervalle betrachtet, wodurch auch für Funktionen mit mehreren Variablen nur ein Funktionswert zu Beginn berechnet werden muss. Aus diesen Überlegungen heraus müssen die Schranken aus (3.2) wie folgt für den Mittelpunkt $c = \frac{a+b}{2}$ des Intervalls $[a, b]$ angepasst werden:

$$\begin{aligned} f(x) &\geq f(c) - K(x - c), \ \forall x \geq c \\ f(x) &\geq f(c) + K(x - c), \ \forall x \leq c \end{aligned} \tag{3.6}$$

Die Änderung der Schranken gegenüber dem Shubert-Verfahren kann in Abb. 3.4 nachvollzogen werden.

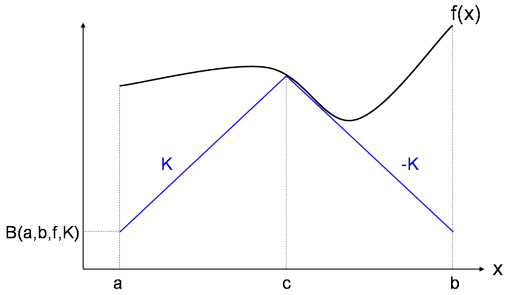


Abb. 3.4: Untere Schranke nach dem DiRect-Verfahren

Wie man sieht, muss auch die Berechnungsvorschrift für den kleinsten Wert B der Schranke (3.6) geändert werden. Sie wird dann zu:

$$B(a, b, f, K) = f(c) - \frac{K(b - a)}{2} \qquad \text{mit } \ c = \frac{a + b}{2} \tag{3.7}$$

Um zu gewährleisten, dass c auch nach der Teilung des Intervalls $[a, b]$ wieder Mittelpunkt eines Teilintervalls ist, wird der Bereich nicht in zwei, sondern in drei gleichlange Intervalle, wie in Abb. 3.5 dargestellt, aufgeteilt.

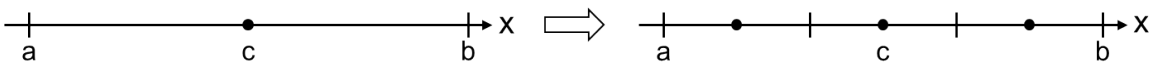
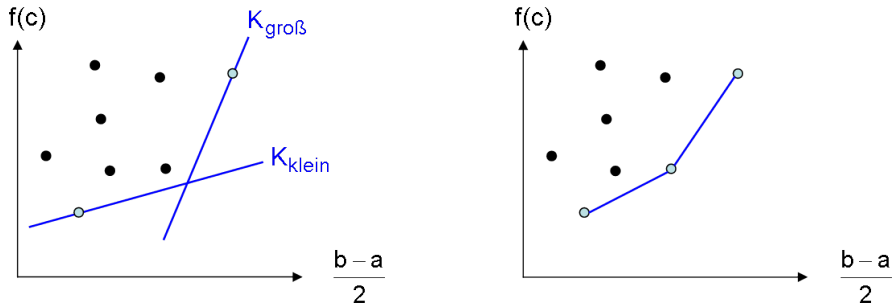


Abb. 3.5: 3-Teilung eines Intervalls

Das zweite Problem des Shubert-Algorithmus besteht darin, dass die Lipschitz-Konstante in der Regel nicht bekannt ist oder nur sehr grob abgeschätzt werden kann. Deswegen wird beim DiRect-Verfahren für die Auswahl des nächsten zu teilenden Intervalls auf diese konkrete Konstante verzichtet und es werden stattdessen alle möglichen Lipschitz-Konstanten verwendet. In Abb. 3.6



(a) Shubert-Verfahren (b) DiRect-Verfahren

Abb. 3.6: Einfluss der Konstante K auf die Wahl des nächsten Teilungspunktes

sind beispielhaft die Mittelpunkte der Teilintervalle, die bis dahin in einem Algorithmus entstanden sind, in Abhängigkeit vom Funktionswert und der Intervallgröße dargestellt. In dieser Abbildung lässt sich der Einfluss der Konstante K auf die Wahl des nächsten Teilungspunktes (hellblau) erkennen. Nach der Methode von Shubert in Abb. 3.6a wird bei einem großen K der Mittelpunkt eines großen Intervalls gewählt (globale Suche), für kleine K fällt die Wahl auf einen Punkt mit kleinem Funktionswert (lokale Suche). Im Gegensatz dazu werden im DiRect-Verfahren mit Hilfe unterschiedlicher K-Werte mehrere Punkte ausgewählt, die sowohl aus einer lokalen als auch einer globalen Sicht stammen. Die zu den ausgewählten Mittelpunkten gehörenden Intervalle heißen *potentiell optimal* und entsprechen einem Teil der konvexen Hülle aller Teilintervalle (siehe Abb. 3.6b). Die Mittelpunkte der potentiell optimalen Intervalle sind in [16] wie folgt definiert:

Definition 3.2: Mittelpunkt eines potentiell optimalen Intervalls
Angenommen, das Intervall $[l, u]$ wurde in m Teilintervalle $[a_i, b_i],\ i = 1, 2, ..., m$, aufgespaltet. Außerdem sei eine Konstante $\epsilon > 0$ und das aktuelle Minimum f_{min} der Funktion $f(x)$ gegeben. Dann ist c_j *Mittelpunkt eines potentiell optimalen Intervalls* j, wenn es ein $\widetilde{K} > 0$ gibt, so dass gilt:

$$f(c_j) - \frac{\widetilde{K}(b_j - a_j)}{2} \leq f(c_i) - \frac{\widetilde{K}(b_i - a_i)}{2},\ \forall i = 1, 2, ..., m \tag{3.8}$$

$$\text{und} \quad f(c_j) - \frac{\widetilde{K}(b_j - a_j)}{2} \leq f_{min} - \epsilon|f_{min}| \tag{3.9}$$

Durch die Forderung (3.8) wird das Intervall mit dem kleinsten Wert für B aus (3.7) bzgl. der Konstanten $\widetilde{K}$ ausgewählt. Durch Variation von $\widetilde{K}$ erhält man somit unterschiedliche Mittelpunkte. Mit Hilfe der Forderung (3.9) werden zu kleine Intervalle von den Betrachtungen ausgeschlossen, so dass eine zu lokale Suche verhindert wird. Durch die Konstante ϵ wird dabei festgelegt, welche Intervalle als zu klein gelten. Ein guter Wert für ϵ ist nach Finkel 10^{-4} (siehe [8]).

Für Funktionen, die von mehreren Variablen abhängen, besteht der zulässige Bereich nicht aus einem Intervall, sondern aus einem Hyper-Rechteck[1]. Die Forderungen aus Definition 3.2 müssen

[1]Ein Hyper-Rechteck der Dimension n ist das Kreuzprodukt von n Intervallen.

wie folgt für diese Hyper-Rechtecke angepasst werden:

$$f(c_j) - \widetilde{K} d_j \leq f(c_i) - \widetilde{K} d_i, \ \forall i = 1, 2, ..., m \tag{3.10}$$

$$\text{und} \quad f(c_j) - \widetilde{K} d_j \leq f_{min} - \epsilon |f_{min}| \tag{3.11}$$

wobei d_j ein Maß für die Hyper-Rechtecke ist. Zum Beispiel kann für d_j der Abstand des Mittelpunktes c_j zu den jeweiligen Eckpunkten gewählt werden.
Eine effiziente Umsetzung dieser Forderungen für einen Algorithmus liefert Lemma 3.1 aus [9].

Lemma 3.1:
Gegeben sei eine Konstante $\epsilon > 0$ und das aktuelle Minimum f_{min} der Funktion $f(x)$. Außerdem sei I die Menge der Indizes aller bisher entstandenen Hyper-Rechtecke. Die Hyper-Rechtecke können dann für ein festes $j \in I$ wie folgt aufgeteilt werden:

$$\begin{aligned} I_1 &= \{i \in I : d_i < d_j\} \\ I_2 &= \{i \in I : d_i > d_j\} \\ I_3 &= \{i \in I : d_i = d_j\} \end{aligned}$$

Das Hyper-Rechteck j ist potentiell optimal wenn die folgenden drei Bedingungen gelten:

- $f(c_j) \leq f(c_i), \ \forall i \in I_3$
- Es existiert ein $\widetilde{K} > 0$, so dass

$$\max_{i \in I_1} \frac{f(c_j) - f(c_i)}{d_j - d_i} \leq \widetilde{K} \leq \min_{i \in I_2} \frac{f(c_i) - f(c_j)}{d_i - d_j}$$

- Und es gilt

$$\begin{aligned} \epsilon &\leq \frac{f_{min} - f(c_j)}{|f_{min}|} + \frac{d_j}{|f_{min}|} \min_{i \in I_2} \frac{f(c_i) - f(c_j)}{d_i - d_j}, \ f_{min} \neq 0 \\ \text{bzw.} \quad f(c_j) &\leq d_j \min_{i \in I_2} \frac{f(c_i) - f(c_j)}{d_i - d_j}, \ f_{min} = 0 \end{aligned}$$

Der Beweis für dieses Lemma kann in [9] nachgelesen werden.

Nun kann das DiRect-Verfahren wie in Algorithmus 3.2 formuliert werden.

Erläuterung: zu Algorithmus 3.2
In Zeile 1 wird der zulässige Bereich, welcher ein Hyper-Rechteck ist, auf einen Hyper-Würfel[1] normiert, wodurch die weiteren Berechnungen vereinfacht werden. Die potentiell optimalen Hyper-Rechtecke werden nach Lemma 3.1 bestimmt, wobei die zweite Forderung in

$$\max_{i \in I_1} \frac{f(c_j) - f(c_i)}{d_j - d_i} - \min_{i \in I_2} \frac{f(c_i) - f(c_j)}{d_i - d_j} \leq 0$$

[1] Ein Hyper-Würfel der Dimension n ist in Analogie zu Hyper-Rechtecken das Kreuzprodukt von n Einheitsintervallen.

Algorithmus 3.2: Dividing-Rectangles-Verfahren

Input: EOA mit Box-Restriktionen, Konstante $\epsilon > 0$

Output: lokales Minimum der Aufgabe

1: Normiere den zulässigen Bereich auf einen Hyper-Würfel
2: Berechne den Funktionswert des Mittelpunktes des Hyper-Würfels und setze f_{min} auf diesen Wert
3: **while** Abbruchbedingungen nicht erfüllt **do**
4: Bestimme die Menge der potentiell optimalen Hyper-Rechtecke und deren Mittelpunkte nach Lemma 3.1
5: Teile diese Hyper-Rechtecke entlang ihrer längsten Seite
6: **end while**

umgeformt wird, wodurch die Konstante $\widetilde{K}$ nicht mehr benötigt wird. Durch die Teilung entlang der längsten Seite entsteht eine effektivere Suche, da der Wertebereich nicht in schmale Hyper-Rechtecke zerfällt, sondern gleichmäßig geteilt wird.
Als Abbruchbedingung für diesen Algorithmus kann eine maximale Anzahl an Funktionsauswertungen, Teilungen oder Iterationen gewählt werden.

Die Abb. 3.7 zeigt eine mögliche Teilung eines zweidimensionalen zulässigen Bereiches, wobei die potentiell optimalen (Hyper-)Rechtecke blau unterlegt sind.

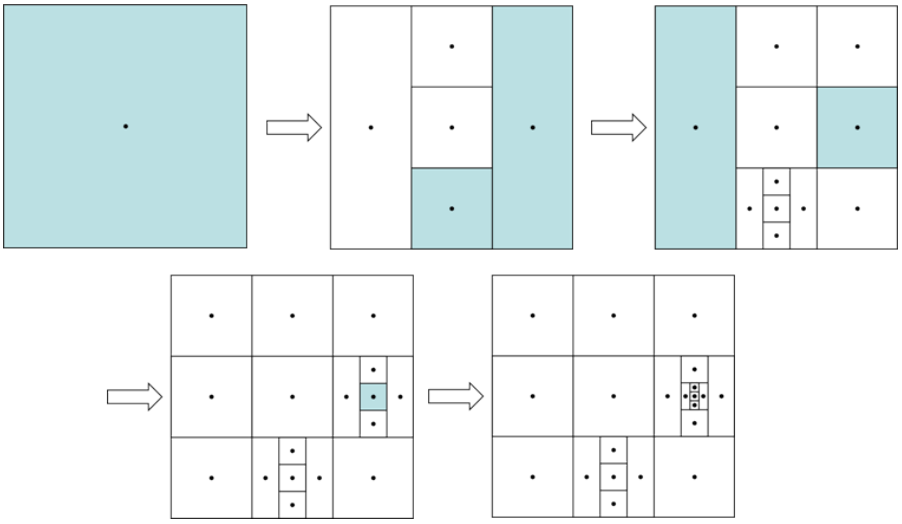


Abb. 3.7: Mögliche Teilung eines zulässigen Bereiches

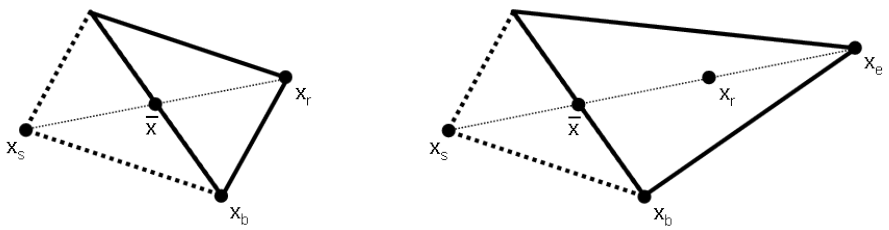
Ist die Zielfunktion im Bereich des globalen Optimums stetig, so kann bewiesen werden, dass das Verfahren gegen das globale Optimum konvergiert (vgl. [16]). Ein weiterer Vorteil ist die Ausgewogenheit zwischen lokaler und globaler Suche sowie die wenigen benötigten Eingangsdaten, wodurch die DiRect-Methode ein effektives Verfahren zur Lösung Box-restringierter EOA darstellt. Andere Restriktionen als Box-Restriktionen können mit Hilfe des *Penalty-Ansatzes*, bei dem jede Verletzung einer Restriktion mit einem möglichst großen Wert in der Zielfunktion bestraft wird,

bewältigt werden. Mit diesem Strafwert wird somit verhindert, dass ein unzulässiger Punkt als gute Lösung angenommen wird. Sind keine Box-Restriktionen für eine Variable gegeben, so wird die Variable mit einem Intervall $[M_1, M_2]$ eingeschränkt, so dass der zulässige Bereich tatsächlich ein Hyper-Rechteck ist. Dabei müssen die Grenzen jedoch so gewählt werden, dass die optimale Lösung innerhalb dieses Intervalls liegt. MOA können nur unter Zuhilfenahme der klassischen Lösungsmethoden, die die Aufgaben in EOA überführen, gelöst werden (siehe Abschnitt 2.3.2). Das Verfahren eignet sich daher weniger für MOA und Aufgaben mit Nebenbedingungen, welche nicht als Box-Restriktionen formuliert sind.

3.2.2 Nelder-Mead-Verfahren

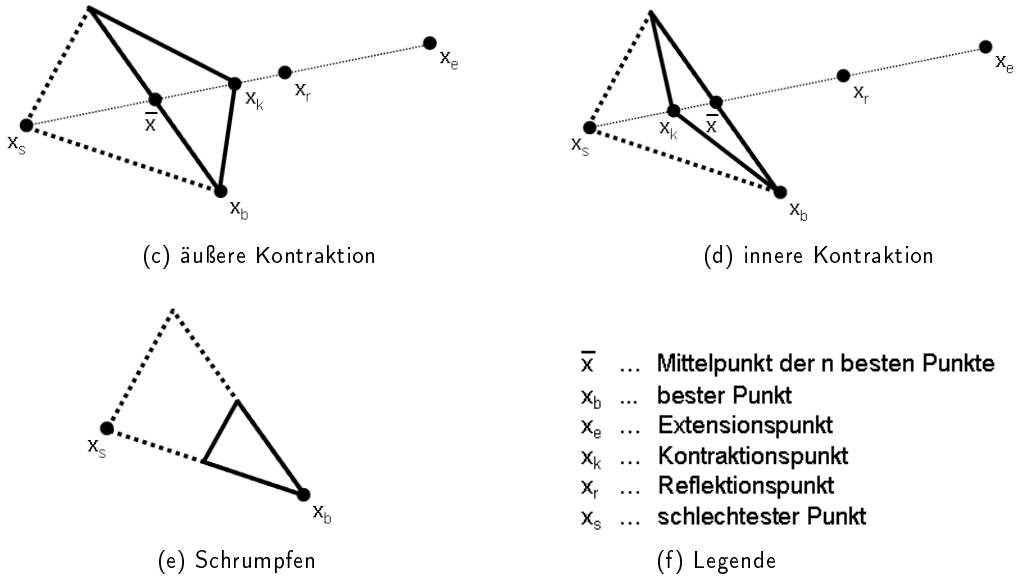
Die nun vorgestellte Methode wurde 1965 von Nelder und Mead entwickelt und wird u.a. in [5], [10], [19] und [29] beschrieben. Man findet sie auch unter dem Namen *(Downhill-)Simplex-Verfahren*. Ausgangspunkt des Nelder-Mead-Verfahrens ist ein $(n+1)$-eckiges Polyeder, das so genannte *Simplex*. Dabei bezeichnet n die Anzahl der Variablen und die Eckpunkte des Simplex stehen für mögliche Lösungen der Aufgabe. Man beginnt also im Gegensatz zu vielen anderen Verfahren nicht nur mit einer, sondern mit $n+1$ Startlösungen. Die Idee des Verfahrens besteht nun darin, den schlechtesten Eckpunkt des Simplex durch einen besseren zu ersetzen, wodurch ein neues Simplex entsteht. Für die Bestimmung des neuen Punktes werden die folgenden Operationen, welche in Abb. 3.8 für eine OA mit zwei Variablen veranschaulicht sind, genutzt:

- Reflektion am Mittelpunkt der n besten Punkte
- Extension des reflektierten Punktes
- innere und äußere Kontraktion des reflektierten Punktes
- Schrumpfen des gesamten Simplex um den besten Punkt



(a) Reflektion (b) Extension

Abb. 3.8: Operationen zur Bestimmung eines neuen Simplex



(c) äußere Kontraktion (d) innere Kontraktion

(e) Schrumpfen (f) Legende

Abb. 3.8: Operationen zur Bestimmung eines neuen Simplex

Ein ausführlicher Algorithmus für das Nelder-Mead-Verfahren kann z.B. in [10] nachgelesen werden. Eine Zusammenfassung liefert der folgende Algorithmus 3.3:

Algorithmus 3.3: Nelder-Mead-Verfahren

Input: unrestringierte EOA

Output: lokales Minimum der Aufgabe

1: Bestimme ein Startsimplex
2: **while** Abbruchbedingungen nicht erfüllt **do**
3: Ermittle den Eckpunkt mit dem größten Funktionswert (x_s), dem zweitgrößten Funktionswert (x_z) und dem kleinsten Funktionswert (x_b)
4: Bestimme den Mittelpunkt $\overline{x}$ aller Eckpunkte ohne x_s
5: Reflektiere x_s an $\overline{x}$. Man erhält x_r
6: **if** $f(x_b) \leq f(x_r) \leq f(x_z)$ **then**
7: Ersetze x_s durch x_r
8: **else**
9: Führe eine der Operationen Extension, Kontraktion oder Schrumpfen durch
10: **end if**
11: **end while**

Erläuterung: zu Algorithmus 3.3

Für die else-Anweisung in Zeile 9 gelten folgende Regeln: Ist der Zielfunktionswert des reflektierten Punktes x_r kleiner als $f(x_b)$, so wird geprüft, ob in Richtung der Reflektion eine noch größere Verbesserung zu erreichen ist (Extension). Ist der reflektierte Punkt jedoch schlechter, so ist das Simplex vermutlich zu groß und x_r wird kontrahiert. Ist der Zielfunktionswert des kontrahierten Punktes x_k größer als $f(x_s)$, so ist das Simplex immer noch zu groß und es wird um einen

festgelegten Faktor geschrumpft, wobei der beste Punkt erhalten bleibt. Anschaulich gesprochen verlagert sich das Simplex im Laufe des Verfahrens in Richtung eines lokalen Optimums und zieht sich dann um dieses zusammen.
Der Algorithmus kann abgebrochen werden, wenn sich der Zielfunktionswert des neu berechneten Punktes nicht wesentlich von den anderen Zielfunktionswerten unterscheidet oder eine festgelegte Anzahl an Iterationen durchlaufen wurde.

Eine Erweiterung des Nelder-Mead-Verfahrens ist die *Complex-Strategie* von Box, mit der Nebenbedingungen in Ungleichungsform berücksichtigt werden können (vgl. [21]). Eine weitere Möglichkeit für den Umgang mit Restriktionen liefert der Penalty-Ansatz (siehe 3.2.1). Speziell für Box-Restriktionen kann auch eine *Intervall-Substitution* durchgeführt werden, bei der das zulässige Intervall $[x_i^L, x_i^U]$ für die Variable x_i mittels einer geeigneten Vorschrift auf das Intervall $(-\infty, \infty)$ transformiert wird und die Restriktionen somit entfallen. Eine solche Vorschrift bietet z.B. die Funktion $d : [x_i^L, x_i^U] \rightarrow (-\infty, \infty)$ aus [24] mit:

$$d(x_i^\circ) = ln\left(\frac{x_i^\circ - x_i^L}{x_i^U - x_i^\circ}\right),\ \forall x_i^\circ \in [x_i^L, x_i^U] \tag{3.12}$$

Da das Nelder-Mead-Verfahren ein Verfahren zur Lösung einkriterieller Aufgaben ist, können MOA nur mit Hilfe der klassischen Lösungsmethoden gelöst werden.

Der Algorithmus für das Nelder-Mead-Verfahren lässt sich leicht implementieren und ist sehr effizient für viele praktische Anwendungen. Ein großer Vorteil des Verfahrens besteht darin, dass es keine Ableitungen benötigt und nur auf dem Vergleich von Zielfunktionswerten beruht, wobei schon wenige Funktionsauswertungen genügen, um eine gute Konvergenz und Stabilität für die meisten Probleme zu erhalten. Jedoch konnte für allgemeine Probleme noch kein Konvergenzbeweis geliefert werden und für spezielle Probleme, z.B. die McKinnon-Funktion (siehe [5] und [19]) kann gezeigt werden, dass das Verfahren in einen nicht-optimalen Punkt[1] konvergiert. Außerdem hängt die Konvergenz und die Güte der gefundenen Lösung entscheidend von der Wahl des Startsimplex ab. Um eine möglichst gute Lösung zu erhalten sollte daher der Algorithmus mit verschiedenen Simplexen gestartet werden. Am besten eignet sich das Verfahren nach Nelder und Mead für unrestringierte EOA mit wenigen ($n \leq 10$) Parametern (vgl. [10]).

3.2.3 Partikelschwarmoptimierung

Die Partikelschwarmoptimierung (PSO) wurde 1995 von Kennedy und Eberhart entwickelt und wird neben der Anwendung in der Mathematik auch zur Erkennung von Parkinson und zur Erstellung von Stundenplänen sowie für grafische Anwendungen, z.B. der Visualisierung von dreidimensionalen Bildern, genutzt. Die nachfolgenden Ausführungen zu diesem Verfahren sind an [4], [20] und [24] angelehnt.

[1] Ein nicht-optimaler Punkt ist ein Punkt des zulässigen Bereiches, welcher weder lokal noch global optimal ist.

PSO ist ein naturanaloges stochastisches Verfahren, welches das soziale Verhalten von Vogel- und Fischschwärmen zum Vorbild hat. Die relativ einfachen Individuen eines solchen Schwarms haben in der Regel nur begrenzte Fähigkeiten, der Schwarm als Ganzes hingegen kann komplexe Aufgaben wie die Suche nach Nahrung oder Rastplätzen oder den Nestbau bewältigen. In der Optimierung stellen die verschiedenen Individuen die unterschiedlichen Lösungen der OA dar. Jedes Individuum wird durch ein so genanntes *Partikel* repräsentiert, welches sich durch den Raum der zulässigen Lösungen bewegt und sich dabei sowohl an seiner bislang besten als auch an der global besten Lösung orientiert. Jedes Partikel ist also durch einen Punkt im Raum, an dem es sich zu einem bestimmten Zeitpunkt befindet, und einer Geschwindigkeit zu diesem Zeitpunkt gekennzeichnet. Diese beiden Merkmale werden zu Beginn des Algorithmus für eine fest vorgegebene Anzahl $P \in \mathbb{N}$ an Partikeln zufällig initialisiert und ändern sich im weiteren Verlauf der Optimierung ständig. Die Änderungen werden dabei immer für einen Zeitschritt $t \in \mathbb{N}$ der Länge 1 durchgeführt.

Der klassische Algorithmus für eine PSO zur Lösung einer unrestringierten EOA lässt sich wie folgt darstellen:

Algorithmus 3.4: Partikelschwarmoptimierung

Input: unrestringierte EOA, Anzahl der Partikel P

Output: lokales Minimum der Aufgabe

1: Zufällige Initialisierung aller P Partikel mit einer Position im Raum und einer Geschwindigkeit
2: **while** Abbruchbedingungen nicht erfüllt **do**
3: Berechne Zielfunktionswert für alle Partikel
4: Ermittle global beste Position
5: **for all** Partikel **do**
6: Bestimme persönlich beste Position
7: Berechne neue Geschwindigkeit und neue Position
8: **end for**
9: **end while**

Die Berechnung der neuen Geschwindigkeit eines Partikels s erfolgt nach folgender Formel:

$$v^s(t+1) = v^s(t) + \underbrace{c_1 r_1 (p^{b_s}(t) - p^s(t))}_{kognitiver Anteil} + \underbrace{c_2 r_2 (p^g(t) - p^s(t))}_{sozialer Anteil} \tag{3.13}$$

Dabei sind $v^s(t)$ und $v^s(t+1)$ die aktuelle und die neue Geschwindigkeit, sowie $p^s(t)$ die aktuelle Position des Partikels s. Mit $p^{b_s}(t)$ wird die persönlich beste bisher erreichte Position des Partikels s bezeichnet und $p^g(t)$ ist die global beste Position aller Partikel zum Zeitpunkt t. c_1 und c_2 sind Konstanten und r_1 sowie r_2 gleichverteilte Zufallszahlen aus dem Intervall $(0, 1)$.
Durch den kognitiven Anteil wird das Partikel zu seinem bisher besten Punkt gezogen, der soziale Anteil bewirkt eine Annäherung an die global beste Position. Die Konstanten c_1 und c_2 regeln dabei den Einfluss dieser Anteile. Ist $c_1 > c_2$ führt dies zu einer stärkeren Exploration der Suchraums, d.h. der zulässige Bereich wird weiträumig durchsucht. Wählt man hingegen $c_1 < c_2$, so streben

die Partikel stärker zur derzeit global besten Lösung, was zu einer schnellen, aber möglicherweise lokalen, Konvergenz führt. Durch die Einbeziehung der aktuellen Geschwindigkeit wird verhindert, dass es zu ruckartigen Bewegungen der Partikel kommt, da dies auch in der Natur untypisch ist.
Die neue Position eines Partikels ergibt sich dann wie folgt aus der alten Position und der neuen Geschwindigkeit:

$$p^s(t+1) = p^s(t) + v^s(t+1) \tag{3.14}$$

Die Bewegung eines Partikels für einen Zeitschritt der Länge 1 ist in Abb. 3.9 in Anlehnung an [24] dargestellt.
Der Algorithmus bricht ab, wenn sich alle Partikel nur noch in einem sehr kleinen Bereich bewegen oder wenn eine vorgegebene Anzahl an Durchläufen erreicht ist.

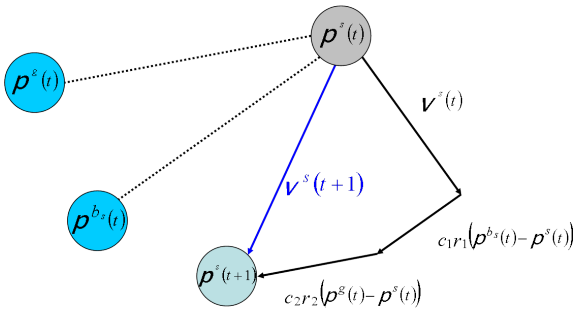


Abb. 3.9: Bewegung eines Partikels

Im Laufe der Jahre wurde der Algorithmus beständig erweitert. So wird der Einfluss der aktuellen Geschwindigkeit im Verlauf der Optimierung gedrosselt, so dass es zu einer schnelleren Konvergenz kommt. Außerdem wird dadurch verhindert, dass Partikel durch eine zu große Geschwindigkeit optimale Lösungen überspringen. Weiterhin wird für den sozialen Anteil nicht die beste Lösung aller Partikel benutzt, sondern nur die in einer bestimmten Nachbarschaft eines Partikels beste Lösung. Die Gleichung (3.13) wird also zu

$$v^s(t+1) = wv^s(t) + c_1 r_1(p^{b_s}(t) - p^s(t)) + c_2 r_2(p^{n_s}(t) - p^s(t)) \tag{3.15}$$

mit w als Drosselungsfaktor und $p^{n_s}(t)$ die beste Position aus der Nachbarschaft des Partikels s.
Für MOA werden nicht die Zielfunktionswerte zur Bestimmung der besten Position genutzt, sondern eine Funktion, die jedem Partikel einen Fitnesswert zuordnet. Der Fitnesswert eines Partikels setzt sich aus seiner Nähe zur Pareto-Front, der Dichte der Partikel zueinander und der Verteilung des Schwarmes im Raum zusammen.
Um Restriktionen einbeziehen zu können, kann der Penalty-Ansatz aus 3.2.1 oder die Intervall-Substitution, wie sie in Abschnitt 3.2.2 beschrieben wurden, genutzt werden. Eine weitere Möglichkeit besteht darin, ein Partikel, welches den zulässigen Bereich verlässt, nach bestimmten Regeln an eine andere, zulässige Position zu setzen.
Für detaillierte Ausführungen zur Berechnung der Fitnesswertfunktion und zum Umgang mit Restriktionen eignet sich besonders [24].

Röber zeigte in [24], dass sich die PSO durch die genannten Erweiterungen sehr gut zum Lösen von EOA mit Nebenbedingungen eignet und es in den meisten Anwendungen zu einer guten und schnellen Konvergenz kommt. Auch für MOA ist dieses Verfahren gut geeignet.

3.2.4 Ersatzmodell-gestützte Verfahren

In diesem Abschnitt sollen nun kurz die Ersatzmodell-gestützten Optimierungsverfahren vorgestellt werden. Die Ausführungen sind dabei an [24], [28] und [30] angelehnt.
Für Optimierungsaufgaben mit zeitaufwändigen Zielfunktionsauswertungen eignen sich die zuvor vorgestellten Verfahren weniger, da sie eine große Anzahl an Funktionswertbestimmungen benötigen, um brauchbare Resultate zu erreichen. Daher entstand die Idee, ein Ersatzmodell der Ausgangsaufgabe zu bilden, bei dem der Zielbereich so vereinfacht wird, dass einerseits eine schnelle Funktionsauswertung möglich ist und andererseits der Wertebereich gut approximiert wird. Da der Zielbereich einer OA jedoch meist unbekannt ist, ist es schwierig, ein geeignetes Modell zu finden. Es wurde daher ein möglichst allgemein gültiges Modell, welches für eine breite Klasse von Funktionen robuste und hinreichend gute Ergebnisse liefert, gesucht. Ein solches Ersatzmodell ist z.B. das *Kriging-Modell*, welches auch in der *EngineeringToolbox* eingesetzt wird und in [28] ausführlich beschrieben ist.

Zu Beginn eines Ersatzmodell-gestützten Optimierungsverfahrens wird mit wenigen, gut verteilten Punkten ein Ersatzmodell initialisiert, welches im weiteren Verlauf mit neuen Punkten trainiert wird. Eine effektive Initialisierung erfolgt nach Stöcker mit Hilfe der Latin-Hypercubes-Methode (vgl. [30]). Die neuen Punkte werden nach so genannten *Auffüllkriterien*[1] ausgewählt. Diese Kriterien wählen entweder Punkte mit einem kleinen Funktionswert aus oder Punkte, die in einem wenig erforschten Gebiet liegen, d.h. in dem die Unsicherheit des Modells hoch ist. Es werden somit sowohl Punkte ausgewählt, die den Zielfunktionswert verbessern, als auch Punkte, die die Genauigkeit des Modells steigern. Die Suche nach neuen Punkten und auch das Training des Ersatzmodells sind dabei eigene Optimierungsaufgaben, welche mit einem beliebigen Verfahren gelöst werden können.
Das Ersatzmodell-gestützte Verfahren wird abgebrochen, wenn eine vorgegeben Anzahl an Funktionsauswertungen erreicht ist oder das Modell die Ausgangsaufgabe hinreichend gut approximiert. Eine Zusammenfassung des Ablaufs liefert Algorithmus 3.5.

Zwei Vertreter der Ersatzmodell-gestützten Optimierungsverfahren, welche in der *EngineeringToolbox* implementiert wurden, sind die Verfahren *Model Assisted Evolutionary Strategy* (MAES) und *Enhanced Modelbased Multiobjective Optimization Algorithm* (EMMOA). Für eine ausführliche Beschreibung dieser Verfahren empfiehlt sich [30].
Das Verfahren MAES kann nur EOA lösen, für MOA müssen daher die Zielfunktionen zu einer

[1] Die verschiedenen Auffüllkriterien werden u.a. in [24] und [28] beschrieben.

Algorithmus 3.5: Ersatzmodell-gestütztes Optimierungsverfahren

Input: OA

Output: lokales Minimum bzw. Approximation der Pareto-Front der Aufgabe

1: Zufällige Initialisierung gleichverteilter Punkte
2: Berechne Zielfunktionswerte der Startpunkte
3: **while** Abbruchbedingungen nicht erfüllt **do**
4: Modelltraining mit vorhandenen Punkten
5: Ermittle neue Punkte mit Hilfe eines Auffüllkriteriums
6: Berechne Zielfunktionswerte der neuen Punkte
7: Aufnahme der neuen Punkte ins Modell
8: **end while**

eindimensionalen Fitnesswertfunktion mittels der verbesserten Tchebyschefffunktion (vgl. [30]) zusammengefasst werden. Im EMMOA wird das Modelltraining für MOA in zwei Phasen unterteilt. Zunächst wird in der so genannten *Tiefensuche* für jede Zielfunktion ein eigenes Modell aufgestellt, welches trainiert wird. In der zweiten Phase, der *Breitensuche*, wird ein neues Modell mit den optimalen Punkten aus der Tiefensuche initialisiert und unter Verwendung einer Fitnesswertfunktion, wie sie in der PSO eingesetzt wird, trainiert. Für EOA wird in der Tiefensuche nur ein Modell für die Zielfunktion aufgestellt und die Breitensuche entfällt, da sie mit der Tiefensuche identisch wäre.

Die Ersatzmodell-gestützten Optimierungsverfahren eignen sich besonders für Aufgaben, deren Zielfunktionsauswertungen aufwändig sind, da sie im Vergleich zu anderen Verfahren in deutlich kürzerer Zeit gute Ergebnisse liefern. Können die Zielfunktionswerte jedoch schnell berechnet werden, so kann es durch die aufwändige Modellbildung zu erheblich größeren Rechenzeiten gegenüber deterministischen oder anderen stochastischen Algorithmen kommen.

4 Diskretisierung der Optimierungsverfahren

Dieses Kapitel beschäftigt sich mit der Diskretisierung der in 3.2 vorgestellten Optimierungsverfahren *Dividing-Rectangles*, *Nelder-Mead*, *Partikelschwarmoptimierung* und des Ersatzmodell-gestützten Optimierungsverfahren *EMMOA*. Im ersten Abschnitt wird die allgemeine Vorgehensweise der Diskretisierung beschrieben. Der zweite Teil beschäftigt sich mit der C++ Klasse *CDiscreteHandler*, mit deren Hilfe stetige Werte diskretisiert werden können. Im letzten Abschnitt wird die Diskretisierung der vier Optimierungsverfahren näher erläutert.

4.1 Allgemeine Vorgehensweise

Wie in Abschnitt 2.4 beschrieben, ist ein Parameter diskret, wenn er nur Werte aus einer endlichen oder abzählbar unendlichen Menge annimmt. Zu diesen Mengen gehören u.a. die Menge der natürlichen, ganzen oder rationalen Zahlen und beliebige Teilmengen dieser Mengen.
In der *EngineeringToolbox* stehen zwei verschiedene Diskretisierungen eines Parameters $x,\ x \in [x_L, x_U]$, zur Verfügung. Zum einen besteht die Möglichkeit, beginnend von der unteren Grenze x_L, mit einer festgelegten Schrittweite s die zulässigen Werte zu erzeugen. Der Parameter nimmt dann nur Werte der Form

$$x = x_L + is,\ i \leq \frac{x_U - x_L}{s},\ i \in \mathbb{N} \tag{4.1}$$

an. Die obere Grenze x_U muss dabei nicht immer erreicht werden.
Die zweite Möglichkeit zur Diskretisierung einer Variable ergibt sich aus der Vorgabe einer Menge zulässiger diskreter Werte, z.B. der Menge der natürlichen Zahlen oder einer Menge von Kosten $\{3.00\,€, 3.01\,€, ..., 3.99\,€, 4.00\,€\}$.

Das allgemeine Prinzip der Diskretisierung eines Optimierungsverfahrens besteht darin, die bei der Berechnung der Zwischenergebnisse entstandenen stetigen Werte geeignet zu diskretisieren, d.h. die stetigen Ergebnisse werden durch die Werte der zulässigen diskreten Menge ersetzt. Dazu muss jedoch überlegt werden, welche Zwischenergebnisse wirklich zu diskretisieren sind und welche stetig bleiben können, damit das Verfahren die gewünschte Genauigkeit und Konvergenz behält. Für die Diskretisierung der Werte wird die in der *EngineeringToolbox* vorhandene Klasse *CDiscreteHandler*, welche in Abschnitt 4.2 beschrieben wird, benutzt. Mit dieser Klasse kann für beide Diskretisierungsarten einem stetigen Wert x_S der zugehörige diskrete Wert x_D zugeordnet werden.
Ein häufiges Problem, welches bei der Diskretisierung mehrerer stetiger Werte auftritt, besteht

darin, dass verschiedene stetige Werte auf denselben diskreten Wert führen. So würde für einen Parameter, dessen Wertebereich die natürlichen Zahlen sind, eine Diskretisierung der Zahlen 1.7 und 2.4 denselben diskreten Wert 2 zurückgeben. In den Verfahren entstehen daher bei der Berechnung der Zwischenergebnisse oft gleiche Zwischenlösungen, deren Funktionswerte im Anschluss einzeln bestimmt werden. Da die Funktionsauswertung bei technischen Aufgaben oft sehr aufwändig ist, ist es daher sinnvoll, nach einer Diskretisierung zu prüfen, ob mehrere gleiche Lösungen entstanden sind. Ist dies der Fall, sollte der Funktionswert dieser Lösung nur einmal berechnet und danach auf alle gleichen Lösungen übertragen werden. Es muss dabei beachtet werden, dass die neu ermittelten Werte sowohl untereinander als auch mit den bisher berechneten Lösungen verglichen werden müssen.

4.2 Die C++ Klasse *CDiscreteHandler*

Diese Klasse ist Bestandteil der *EngineeringToolbox* und wird u.a. zur Initialisierung der Diskretisierungsart und zur Diskretisierung stetiger Werte genutzt. Alle Funktionen, die in dieser Klasse enthalten sind, sind mit einer Kurzbeschreibung in Tabelle 4.1 aufgeführt. Dabei sei angemerkt, dass die diskrete zulässige Menge, welche durch eine Schrittweite oder durch Vorgabe entsteht, intern als aufsteigend sortiertes Feld verwaltet wird.

Funktion	Beschreibung
double Value(int i)	Gibt den *i*-ten Wert des diskreten Feldes zurück
int Count()	Gibt die Mächtigkeit des diskreten Feldes zurück
int Position(double dVal)	Gibt die Stelle des Wertes *dVal* im diskreten Feld zurück
EDiscreteType Type()	Gibt die Diskretisierungsart der Variable zurück
double Discretize(double dVal)	Gibt den diskreten Wert zu *dVal* zurück
bool IsIn(double dVal)	Gibt an, ob *dVal* zum diskreten Feld gehört
bool IsDiscrete()	Gibt an, ob die Variable diskret oder stetig ist

Tab. 4.1: Funktionen der Klasse *CDiscreteHandler*

Zusätzlich zu diesen Funktionen gibt es für jede Diskretisierungsart eine Funktion zur Initialisierung. Je nach Art der Diskretisierung müssen dabei unterschiedliche Werte übergeben werden. So müssen bei einer Diskretisierung mit einer Schrittweite die Grenzen x_L und x_U sowie die Schrittweite s übergeben werden. Für eine Diskretisierung mit vorgegebener Menge muss diese zur Initialisierung übergeben werden.

Soll nun innerhalb eines Optimierungsalgorithmus eine Diskretisierung durchgeführt werden, muss zunächst die Diskretisierungsart einmalig initialisiert werden. Danach kann mit der Funktion *IsDiscrete()* ermittelt werden, welche Parameter zu diskretisieren sind und welche stetig bleiben. Mittels

der Funktion *Discretize()* werden die Werte der diskreten Parameter im Anschluss diskretisiert. Diese Funktion wurde im Rahmen dieser Arbeit neu geschrieben und nutzt je nach Diskretisierungsart Algorithmus 4.1 oder 4.2. Der diskrete Wert wird am Ende an den Optimierungsalgorithmus zurückgegeben und kann dort verwendet werden.

Algorithmus 4.1: Diskretisierung mit einer vorgegebenen Schrittweite

Input: stetiger Wert x_S, Grenzen des Wertebereichs x_L und x_U, Schrittweite s

Output: zu x_S gehöriger diskreter Wert x_D

1: **if** $x_S \leq x_L$ **then**
2: **return** x_L
3: **else**
4: Teile x_S durch s, runde den Wert ab und multipliziere ihn wieder mit s. Man erhält x_D mit $x_D = \left\lfloor \frac{x_S}{s} \right\rfloor * s$
5: **end if**
6: **while** $x_D > x_U$ **do**
7: Ziehe s von x_D ab
8: **end while**
9: **return** x_D

Algorithmus 4.2: Diskretisierung mit einer vorgegebenen diskreten Menge

Input: stetiger Wert x_S, diskrete Menge X_D

Output: zu x_S gehöriger diskreter Wert x_D

1: **if** $x_S \leq min\{X_D\}$ **then**
2: **return** $min\{X_D\}$
3: **else if** $x_S \geq max\{X_D\}$ **then**
4: **return** $max\{X_D\}$
5: **else** /**Suche den Wert x_D, welcher für alle Werte aus X_D den kleinsten Abstand zu x_S hat, d.h. $x_D = arg\left\{\min_{x \in X_D} |x - x_S|\right\}$*/*
6: Setze $d = max\{X_D\} - min\{X_D\}$ /**Initialisierung mit größtmöglichem Wert*/*
7: **for all** Werte x aus X_D **do**
8: **if** $|x - x_S| < d$ **then**
9: Setze $d = |x - x_S|$
10: Setze $x_D = x$
11: **end if**
12: **end for**
13: **end if**
14: **return** x_D

Sind alle Parameter einer OA diskret, so kann über die Funktion *Count()* ermittelt werden, wie viele verschiedene Lösungen maximal berechnet werden können. Dazu müssen die Werte, welche

von *Count()* zurückgegeben werden, miteinander multipliziert werden. Man erhält dadurch die Mächtigkeit der zulässigen Menge der OA. Dieser Wert kann als zusätzliche Abbruchbedingung in jedes Optimierungsverfahren eingebaut werden, denn ist die Anzahl der im Algorithmus ermittelten Lösungen gleich der Anzahl der zulässigen Lösungen, so können im weiteren Verlauf des Verfahrens keine neuen Werte gefunden werden und der Algorithmus kann beendet werden.

Mit der Klasse *CDiscreteHandler* kann nun ein beliebiges Verfahren diskretisiert werden. Die Diskretisierung der vier Optimierungsverfahren *Dividing-Rectangles*, *Nelder-Mead*, *Partikelschwarmoptimierung* und *EMMOA* wird im folgendem Abschnitt näher beschrieben.

4.3 Diskretisierung spezieller Optimierungsverfahren

In diesem Abschnitt werden die Diskretisierungen der einzelnen Optimierungsverfahren, welche im Rahmen dieser Bachelorarbeit neu in die *EngineeringToolbox* integriert wurden, genauer beschrieben. Es wird dabei auf Besonderheiten für die in Abschnitt 3.2 dargelegten Verfahren und die in Abschnitt 4.1 genannten Probleme eingegangen.

4.3.1 Dividing-Rectangles-Verfahren

Das DiRect-Verfahren beginnt, wie in Algorithmus 3.2 in Abschnitt 3.2.1 beschrieben, damit, den Wertebereich $[x_i^L, x_i^U]$ für jeden Parameter $x_i,\ i = 1, 2, ..., n$, auf das Intervall $[0, 1]$ zu normieren. Es muss daher auch die Diskretisierung für alle diskreten Parameter angepasst werden. Das heißt, für einen Parameter $x \in [x_L, x_U]$ mit einer vorgegebenen Schrittweite s muss diese durch

$$s^{(normiert)} = \frac{s}{x_U - x_L}$$

ersetzt werden. Wird der Parameter durch eine vorgegeben Menge X_D diskretisiert, so müssen alle Elemente $x_D \in X_D$ ebenfalls auf das Intervall $[0, 1]$ normiert werden, d.h. die Werte x_D werden zu

$$x_D^{(normiert)} = \frac{x_D - x_L}{x_U - x_L},\ \forall x_D \in X_D$$

Mit dieser Transformation der Diskretisierung kann die Klasse *CDiscreteHandler* initialisiert werden.

Die Diskretisierung der Werte erfolgt zu Beginn des Algorithmus für den Mittelpunkt des Hyper-Würfels und bei der Berechnung der neuen Punkte, welche die Mittelpunkte der neuen Hyper-Rechtecke bilden (vgl. Algorithmus 3.2). Bei der Diskretisierung können jedoch zwei unterschiedliche Mittelpunkte zu einer Lösung zusammen fallen, wie es in Abbildung 4.1 dargestellt ist. In dieser Abbildung ist der zulässige Bereich $[0, 1] \times [0, 1]$ für zwei Variable x_1, x_2 nach der ersten Teilung dargestellt. Beide Parameter sind durch eine Schrittweite $s = \frac{1}{6}$ diskretisiert. Bei der Teilung der hellblau unterlegten potentiell optimalen Rechtecke entstehen zunächst die beiden blauen

Punkte $(\frac{1}{2}, \frac{11}{18})$ und $(\frac{1}{2}, \frac{13}{18})$, welche jedoch nach der Diskretisierung zu einem Mittelpunkt $(\frac{1}{2}, \frac{2}{3})$ (rot) zusammenfallen.

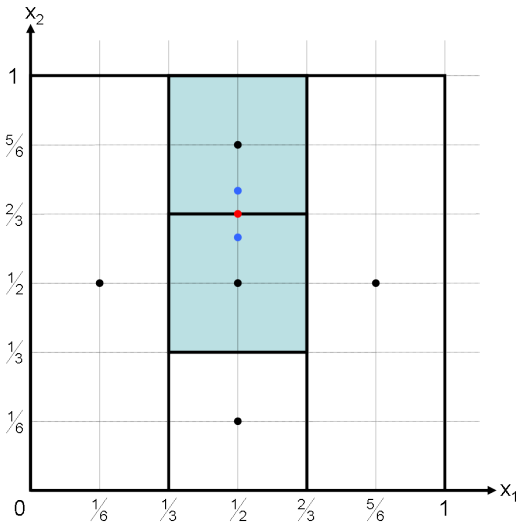


Abb. 4.1: Entstehung gleicher Mittelpunkte

Daher muss vor der Funktionsauswertung der neu berechneten Punkte darauf geachtet werden, dass gleiche Lösungen nicht einzeln berechnet werden. Stattdessen wird der Funktionswert dieser Lösung einmalig berechnet und der Wert anschließend allen gleichen Lösungen zugeordnet. Im DiRect-Algorithmus werden dabei zunächst die neu berechneten Punkte mit den im bisherigen Verlauf ermittelten Lösungen verglichen. Treten dabei Lösungen auf, deren Funktionswert schon bekannt ist, so können die Funktionswerte den Lösungen zugeordnet werden und die Lösungen aus den weiteren Betrachtungen zur Funktionsauswertung entfernt werden. Anschließend wird geprüft, ob bei der Bestimmung der neuen Mittelpunkte gleiche Lösungen entstanden sind (wie in Abb. 4.1 dargestellt). Ist dies der Fall, so werden die mehrfach auftretenden Lösungen gespeichert, einmalig der Funktionswert einer solchen Lösung bestimmt und anschließend der Wert allen gemerkten Lösungen zugeordnet.

Weiterhin wurde eine zusätzliche Abbruchbedingung, wie sie in 4.2 beschrieben wurde, implementiert. Der nachfolgende Algorithmus 4.3 liefert eine Zusammenfassung für das diskretisierte DiRect-Verfahren.

Algorithmus 4.3: Diskretisiertes Dividing-Rectangles-Verfahren

Input: EOA mit Box-Restriktionen, Konstante $\epsilon > 0$, Schrittweite s bzw. diskrete Menge X_D

Output: lokales Minimum der Aufgabe

1: Normiere den zulässigen Bereich auf einen Hyper-Würfel
2: Initialisiere die C++ Klasse *CDiscreteHandler* mit normierter Schrittweite $s^{(normiert)}$ bzw. diskreter Menge $X_D^{(normiert)}$
3: Diskretisiere den Mittelpunkt des Hyper-Würfels
4: Berechne den Funktionswert des diskretisierten Mittelpunktes und setze f_{min} auf diesen Wert
5: **while** Abbruchbedingungen nicht erfüllt **do**
6: Bestimme die Menge der potentiell optimalen Hyper-Rechtecke und deren Mittelpunkte nach Lemma 3.1
7: Diskretisiere die neu ermittelten Punkte
8: Teile diese Hyper-Rechtecke entlang ihrer längsten Seite
9: **end while**

4.3.2 Nelder-Mead-Verfahren

Die Diskretisierung des Nelder-Mead-Verfahrens erfolgt im Algorithmus 3.3 (siehe Abschnitt 3.2.2) immer dann, wenn ein neuer Punkt mit Hilfe einer der Operationen Reflektion, Extension, Kontraktion oder Schrumpfen bestimmt wird. Außerdem müssen die zufällig erzeugten Starteckpunkte des Ausgangssimplex diskretisiert werden. Somit wird zu jedem Zeitpunkt im Algorithmus mit diskreten Werten gerechnet.
Für die Diskretisierung der neu ermittelten Punkte wurde eine neue Funktion *discretize()* implementiert. Mit dieser Funktion wird eine übergebene Matrix $MatToDisc$ diskretisiert und mit den diskreten Werten überschrieben. Die Zeileneinträge der Matrix entsprechen dabei den verschiedenen Zwischenlösungen, die diskretisiert werden sollen, und die Spalteneinträge stehen für die einzelnen Parameter. Die Diskretisierung erfolgt spaltenweise und ist in Algorithmus 4.4 festgehalten.

Algorithmus 4.4: Die Funktion *discretize()*

Input: Matrix $MatToDisc$

1: **for** $i := 1$ bis Anzahl Parameter **do**
2: **if** Parameter i ist diskret **then**
3: **for** $j := 1$ bis Anzahl Zwischenlösungen **do**
4: Ersetze $MatToDisc(j, i)$ durch diskretisierten Wert
5: **end for**
6: **end if**
7: **end for**

Erläuterung: zu Algorithmus 4.4
Die erste Schleife durchläuft alle Parameter. Dies entspricht einer Schleife über alle Spalten der Matrix $MatToDisc$. In der zweiten Schleife werden alle Zeilen, d.h. alle Zwischenlösungen, der

Matrix durchlaufen, jedoch nur, wenn der Parameter i diskret ist (siehe Zeile 2). Diese Überprüfung kann mit Hilfe der Funktion *IsDiscrete()* der Klasse *CDiscreteHandler* durchgeführt werden. Die Diskretisierung der Matrixwerte in Zeile 4 wird mit der Funktion *Discretize()* aus *CDiscreteHandler* realisiert.

Nun muss noch zu Beginn des Algorithmus die Klasse *CDiscreteHandler* initialisiert werden. Eine Besonderheit ergibt sich dabei, wenn eine Intervall-Substitution, wie sie am Ende von Abschnitt 3.2.2 beschrieben wurde, durchgeführt werden soll. Denn wird der zulässige Bereich $[x_L, x_U]$ einer Variable x auf das Intervall $(-\infty, \infty)$ transformiert, so muss die Schrittweite bzw. die diskrete Menge ebenfalls für das neue Intervall angepasst werden. Da die Transformationsvorschrift nicht linear ist (vgl. Gleichung (3.12)), kann die Schrittweite nicht ohne Weiteres in eine andere Schrittweite überführt werden, wie es beim DiRect-Verfahren in 4.3.1 geschehen ist. Es werden daher die zulässigen Punkte des Intervalls $[x_L, x_U]$ mittels Formel (4.1) ermittelt und anschließend einzeln entsprechend Vorschrift (3.12) transformiert. Man erhält damit eine diskrete zulässige Menge für das Intervall $(-\infty, \infty)$. Mit dieser Menge kann dann die C++ Klasse initialisiert werden. Für eine Diskretisierung mit einer vorgegebenen Menge müssen bei einer Intervall-Substitution die Elemente dieser Menge ebenfalls einzeln mittels der Vorschrift (3.12) transformiert werden und können dann zur Initialisierung verwendet werden. Die Initialisierung der Klasse *CDiscreteHandler* bei einer Intervall-Substitution ist in Algorithmus 4.5 zusammengefasst. Wird keine Intervall-Substitution durchgeführt, so kann die Klasse ohne vorherige Umrechnungen wie in 4.2 beschrieben initialisiert werden.

Algorithmus 4.5: Initialisierung des Klasse *CDiscreteHandler* bei einer Intervallsubstitution

Input: Intervallgrenzen $[x_L, x_U]$, Schrittweite s oder diskrete Menge X_D

1: **if** Schrittweite gegeben **then**
2: Berechne diskrete Menge X_D mit $X_D = \{x_L + is,\ i \leq \frac{x_U - x_L}{s} \text{ und } i \in \mathbb{N}\}$
3: **end if**
4: **for all** $x_D \in X_D$ **do**
5: Ersetze x_D durch $ln\left(\frac{x_D - x_L}{x_U - x_D}\right)$
6: **end for**
7: Initialisiere *CDiscreteHandler* mit der diskreten Menge X_D

Problematisch bei der Diskretisierung mit einer Schrittweite bei einer Intervall-Substitution ist, dass für eine feine Diskretisierung oder bei einem großen Intervall sehr viele diskrete Werte entstehen. Dadurch wird auch die Dimension der diskreten Menge, die in Zeile 2 berechnet wird, sehr groß, was zu Speicherplatzproblemen führen kann.

Das diskretisierte Nelder-Mead-Verfahren ist in Algorithmus 4.6 festgehalten.

Algorithmus 4.6: Diskretisiertes Nelder-Mead-Verfahren

Input: EOA, Schrittweite s bzw. diskrete Menge X_D

Output: lokales Minimum der Aufgabe

1: Initialisiere die C++ Klasse *CDiscreteHandler*
2: Bestimme ein Startsimplex
3: Rufe die Funktion *discretize()* mit den Eckpunkten des Startsimplex auf
4: **while** Abbruchbedingungen nicht erfüllt **do**
5: Ermittle den Eckpunkt mit dem größten Funktionswert (x_s), dem zweitgrößten Funktionswert (x_z) und dem kleinsten Funktionswert (x_b)
6: Bestimme den Mittelpunkt $\overline{x}$ aller Eckpunkte ohne x_s
7: Reflektiere x_s an $\overline{x}$. Man erhält x_r
8: **if** $f(x_b) \leq f(x_r) \leq f(x_z)$ **then**
9: Ersetze x_s durch x_r
10: Diskretisiere x_s mittels Funktion *discretize()*
11: **else**
12: Führe eine der Operationen Extension, Kontraktion oder Schrumpfen durch
13: Diskretisiere die neu ermittelten Punkte mit Hilfe der Funktion *discretize()*
14: **end if**
15: **end while**

4.3.3 Partikelschwarmoptimierung

Für die Diskretisierung der Partikelschwarmoptimierung gibt es zwei verschiedene Ansätze: Zum einen kann die Geschwindigkeit diskretisiert werden, so dass man mit einem diskreten Startwert nur diskrete Positionen erreichen kann, zum anderen kann die neue Position, welche mit Gleichung (3.14) aus Abschnitt 3.2.3 berechnet wird, nachträglich diskretisiert werden. Die erste Variante hat den Nachteil, dass die neue Position, die aus der alten diskreten Position und der diskreten Geschwindigkeit berechnet wird, nicht in jedem Fall der diskretisierten stetigen Position nach (3.14) entspricht. Außerdem ergeben sich Schwierigkeiten bei einer Diskretisierung mit einer vorgegebenen Menge, da in diesem Fall die Addition zweier diskreter zulässiger Werte nicht immer einen diskreten zulässigen Wert ergibt. Daher wurde die Diskretisierung nach dem zweiten Ansatz gewählt.
Die Partikelschwarmoptimierung beginnt damit, Position und Geschwindigkeit der P Partikel mit zufälligen Werten zu belegen (vgl. Algorithmus 3.4). Als erstes müssen daher die Startpositionen der Partikel diskretisiert werden. Dies geschieht mittels der Funktion *Discretize()* der Klasse *CDiscreteHandler*. Weiterhin müssen die neu berechneten Punkte, die aus Gleichung (3.14) entstehen, diskretisiert werden, wofür ebenfalls die Funktion *Discretize()* genutzt wird. Die Partikel haben somit zu jedem Zeitpunkt im Algorithmus eine diskrete, zulässige Position, wodurch auch bei einem vorzeitigen Abbruch der Optimierung eine zulässige Lösung ermittelt werden kann.
Bei der Diskretisierung der neuen Positionen kann es vorkommen, dass zwei Partikel nach der

Diskretisierung die gleiche Position haben, wie es in Abb. 4.2 dargestellt ist. In dieser Darstellung ist der Wertebereich $[0, 20] \times [0, 10]$ für die zwei Variablen x_1 und x_2 abgebildet, wobei x_1 mit der Menge $\{2, 9, 16\}$ und x_2 mit der Schrittweite 2 diskretisiert wurde. Außerdem sind die neuen Positionen der beiden Partikel 1 und 2, die bei einer stetigen Rechnung entstehen, in blau eingezeichnet. Man erkennt, dass in diesem Beispiel die Diskretisierung von $p^1(t+1)$ und $p^2(t+1)$ auf dieselbe Position $p^{diskret}$ (rot) führt.

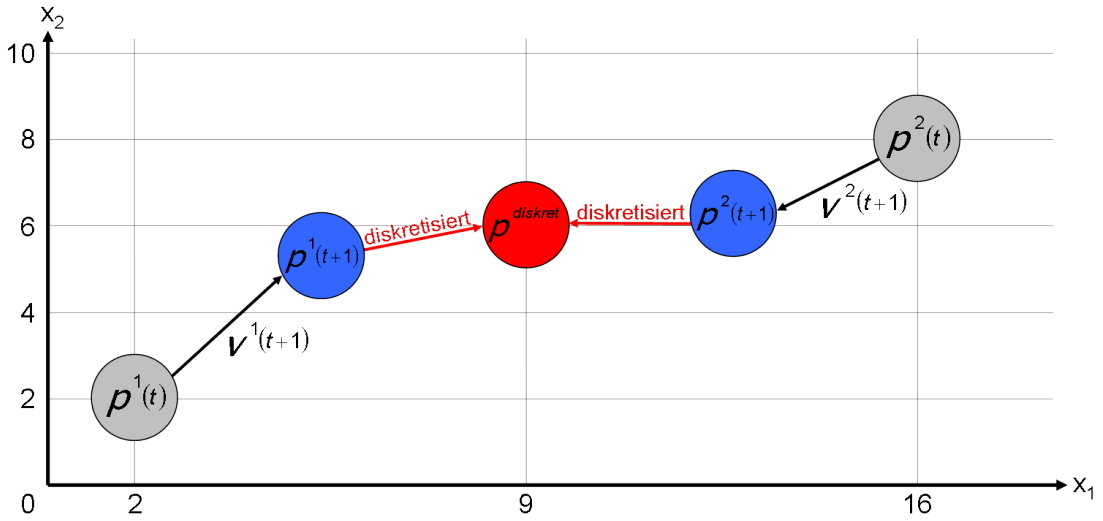


Abb. 4.2: Entstehung gleicher Positionen

Im Algorithmus der Partikelschwarmoptimierung müssen daher die mehrfach entstandenen Positionen gefunden und die Funktionswerte entsprechend zugeordnet werden. Dies geschieht über den gleichen Algorithmus, welcher bereits für das Verfahren DiRect in 4.3.1 beschrieben wurde. Wichtig ist hierbei, dass die Positionen, die mehrfach auftreten, weiterhin einzeln abgespeichert werden, da sich sonst, wie im folgenden Beispiel beschrieben, das Verhalten des Verfahrens ändert.

Beispiel 4.1:
Es werden ein Partikel s und dessen Nachbarschaftspartikel n_1, n_2 und n_3 betrachtet. Angenommen, n_1 und n_2 hätten die selbe Position wie ein weiteres Partikel z, welches nicht zur Nachbarschaft von s gehört, und der Funktionswert dieser Partikel sei viel kleiner als der des Partikels n_3, d.h. $p^{n_1}(t) = p^{n_2}(t) = p^z(t) \neq p^{n_3}(t)$ und $f(n_1) = f(n_2) = f(z) \ll f(n_3)$.
Würden jetzt die Partikel n_1 und n_2 aufgrund der gleichen Position mit z aus dem Schwarm entfernt werden, so wäre das beste Partikel aus der Nachbarschaft von s nicht mehr n_1 oder n_2 sondern n_3. Damit würde sich s in Richtung n_3 bewegen, obwohl dieses Partikel einen großen Funktionswert hat, wodurch sich die Position des Partikels s verschlechtern würde.

Die Diskretisierung der Partikelschwarmoptimierung lässt sich in Algorithmus 4.7 nachvollziehen. Dabei erfolgt die Initialisierung der Klasse *CDiscreteHandler* ebenso wie im Verfahren nach Nelder-Mead (4.3.2), da auch bei der Partikelschwarmoptimierung zur Behandlung von Box-Restriktionen die Möglichkeit einer Intervall-Substitution besteht.

Algorithmus 4.7: Diskretisierte Partikelschwarmoptimierung

Input: EOA, Anzahl der Partikel P, Schrittweite s bzw. diskrete Menge X_D

Output: lokales Minimum der Aufgabe

```
1: Initialisiere die C++ Klasse CDiscreteHandler
2: Zufällige Initialisierung aller P Partikel mit einer Position im Raum und einer Geschwindigkeit
3: Diskretisiere für jedes Partikel die Position
4: while Abbruchbedingungen nicht erfüllt do
5:    Berechne Zielfunktionswert für alle Partikel
6:    Ermittle global beste Position
7:    for all Partikel do
8:       Bestimme persönlich beste Position
9:       Berechne neue Geschwindigkeit und neue Position
10:      Diskretisiere für jedes Partikel die neue Position
11:   end for
12: end while
```

4.3.4 Enhanced Modelbased Multiobjective Optimization Algorithm

Bei der Beschreibung der Diskretisierung der Ersatzmodell-gestützten Optimierungsverfahren wird sich auf das Verfahren Enhanced Modelbased Multiobjective Optimization Algorithm (EMMOA) beschränkt, da andere Verfahren, wie z.B. MAES, in gleicher Weise diskretisiert werden können, jedoch im Rahmen dieser Bachelorarbeit nicht mehr betrachtet werden konnten.

Die Initialisierung der Klasse *CDiscreteHandler* im EMMOA erfolgt wie in 4.2 beschrieben mit Schrittweite sowie Ober- und Untergrenze des Parameters bzw. mit einer vorgegebenen diskreten Menge.
Zu Beginn des Verfahrens werden die zufällig erzeugten Startpunkte, welche dem Modellaufbau dienen (vgl. Algorithmus 3.5), mittels der Funktion *Discretize()* diskretisiert. Im weiteren Verlauf des Algorithmus müssen die neu ermittelten Punkte, die für das Training des Ersatzmodells benötigt werden, diskretisiert werden. Da die Suche nach neuen Punkten in einer eigenständigen Optimierungsaufgabe durchgeführt wird (siehe 3.2.4), wird die Diskretisierung auf diese Aufgabe übertragen, so dass die neu ermittelten Punkte bereits diskretisiert an EMMOA zurückgegeben werden. Für die Suche nach neuen Punkte wird bei EMMOA das Verfahren Ordinary Genetic Algorithm (OGA) genutzt, da dieses bereits über eine Diskretisierung verfügt, welche von Ratzlaff in die *EngineeringToolbox* implementiert und erfolgreich getestet wurde (siehe [23]). Um nun die Diskretisierung korrekt von EMMOA auf OGA zu übertragen, muss die Schrittweite bzw. die diskrete Menge angepasst werden. Die untere bzw. obere Grenze eines diskreten Parameters $x \in [x_L, x_U]$ sei im Verfahren OGA auf x_L^{OGA} bzw. x_U^{OGA} abgebildet. Die zu s gehörige Schrittweite s^{OGA} wird

dann mittels

$$s^{OGA} = \frac{s}{x_U - x_L} * (x_U^{OGA} - x_L^{OGA})$$

ermittelt. Für die Elemente x_D der diskreten Menge X_D aus EMMOA ergibt sich im OGA

$$x_D^{OGA} = \frac{x_D - x_L}{x_U - x_L} * (x_U^{OGA} - x_L^{OGA}) + x_L^{OGA},\ \forall x_D \in X_D$$

Mit dieser transformierten Schrittweite s^{OGA} bzw. Menge X_D^{OGA} kann anschließend das Verfahren OGA initialisiert und ausgeführt werden. Die Werte, die mit diesem Verfahren ermittelt werden, sind dann zulässig für den diskretisierten EMMOA.

Zusätzlich zur Diskretisierung wurde eine weitere Abbruchbedingung in das Verfahren EMMOA implementiert. Mit dieser Bedingung wird der Algorithmus beendet, wenn innerhalb einer bestimmten Zeit keine neuen Punkte gefunden wurden, was bedeutet, dass in einer festen Anzahl an Durchläufen keine neuen Punkte bestimmt werden konnten, mit denen das Ersatzmodell verbessert werden kann. Diese Bedingung verhindert außerdem, dass der Algorithmus in eine Endlosschleife gerät, wenn alle zulässigen diskreten Punkte gefunden wurden.

Der Algorithmus des diskretisierten Verfahrens EMMOA ändert sich im Vergleich zum stetigen Verfahren kaum, da die Diskretisierung der Werte hauptsächlich im Verfahren OGA durchgeführt wird. Eine Zusammenfassung für die Diskretisierung liefert Algorithmus 4.8.

Algorithmus 4.8: Diskretisierter EMMOA

Input: OA, Schrittweite s bzw. diskrete Menge X_D

Output: lokales Minimum bzw. Approximation der Pareto-Front der Aufgabe

1: Initialisiere die C++ Klasse *CDiscreteHandler* für EMMOA mit Schrittweite s bzw. X_D
2: Initialisiere die C++ Klasse *CDiscreteHandler* für OGA mit Schrittweite s^{OGA} bzw. X_D^{OGA}
3: Zufällige Initialisierung gleichverteilter Punkte
4: Diskretisiere diese Startpunkte
5: Berechne Zielfunktionswerte der Startpunkte
6: **while** Abbruchbedingungen nicht erfüllt **do**
7: Modelltraining mit vorhandenen Punkten
8: Ermittle neue Punkte mit Hilfe eines Auffüllkriteriums
9: Berechne Zielfunktionswerte der neuen Punkte
10: Aufnahme der neuen Punkte ins Modell
11: **end while**

5 Bewertung der diskretisierten Optimierungsverfahren

Für die Bewertung der diskretisierten Optimierungsverfahren wurden zwei analytische und eine technische Aufgabe aus dem Bereich der Getriebeoptimierung genutzt. Die Ergebnisse, die mit Hilfe dieser Testaufgaben ermittelt wurden, werden in diesem Kapitel vorgestellt. Dazu werden zunächst die verwendeten Testaufgaben beschrieben. Die Lösungen dieser Aufgaben, die mit den verschiedenen Optimierungsverfahren berechnet wurden, werden im zweiten Abschnitt dargelegt, ausgewertet und miteinander verglichen. Im Anschluss wird eine Bewertung der einzelnen Verfahren geben.

5.1 Testaufgaben

5.1.1 Analytische Aufgaben

Für die Bewertung der Verfahren mit analytischen Aufgaben wurden zwei Testprobleme aus [30] ausgewählt und die dort verwendeten Bezeichnungen beibehalten. Für beide Aufgaben sind die optimalen Lösungen bekannt und können daher genutzt werden, um die Lösungen, die von den in dieser Arbeit diskretisierten Optimierungsverfahren ermittelt wurden, zu bewerten.

Himmelblau

Diese von Himmelblau entworfene Aufgabe besitzt zwei Parameter mit Box-Restriktionen und eine Zielfunktion. Die OA ist in Abb. 5.1 dargestellt und definiert als:

$$\begin{aligned} f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 + 0.1(x_1 - 3)^2(x_2 - 2)^2 \rightarrow min \\ -6 \leq x_i \leq 6,\ i = 1, 2 \\ x_i \in \mathbb{R},\ i = 1, 2 \end{aligned} \tag{5.1}$$

Das globale Minimum liegt bei $x^* = (x_1^*, x_2^*) = (3, 2)$ mit einem Minimalwert $f(x^*) = 0$.

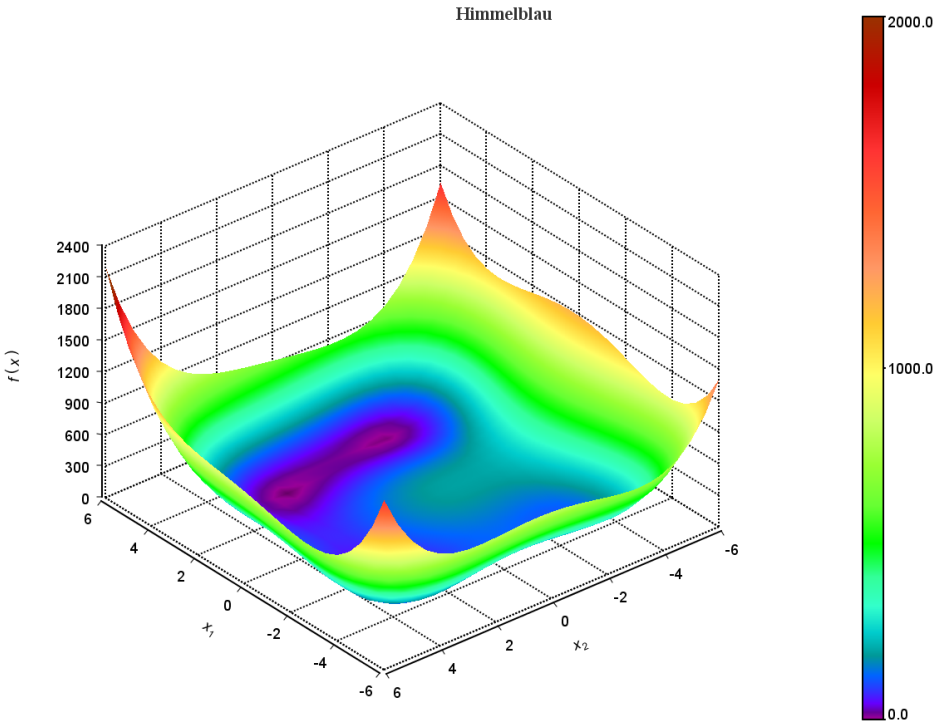


Abb. 5.1: Testfunktion Himmelblau

DTLZ

Diese MOA wurde wie folgt von Deb, Thiele, Laumanns und Zitzler (DTLZ) aufgestellt:

$$
\begin{aligned}
f_1(x) &= \frac{1}{2}x_1(1+s(x)) \rightarrow min \\
f_2(x) &= \frac{1}{2}(1-x_1)(1+s(x)) \rightarrow min \\
\text{mit } s(x) &= 100\left(5+\sum_{i=2}^{6}\left((x_i-0.5)^2-\cos(2\pi(x_i-0.5))\right)\right) \\
& \quad 0 \leq x_i \leq 1,\ i=1,2,...,6 \\
& \quad x_i \in \mathbb{R},\ i=1,2,...,6
\end{aligned} \tag{5.2}
$$

Die beiden Zielfunktionen sind in Abb. 5.2 dargestellt ($f_1(x)$ mit linker, $f_2(x)$ mit rechter Farbskala). Die effizienten Punkte dieser Aufgabe erhält man für $x_i = 0.5,\ i = 2,3,...,6$, und x_1 beliebig aus $[0,1]$. Die Pareto-Menge hat folglich die Form $P = \{(x_1, 0.5, 0.5, 0.5, 0.5, 0.5) | x_1 \in [0,1]\}$.

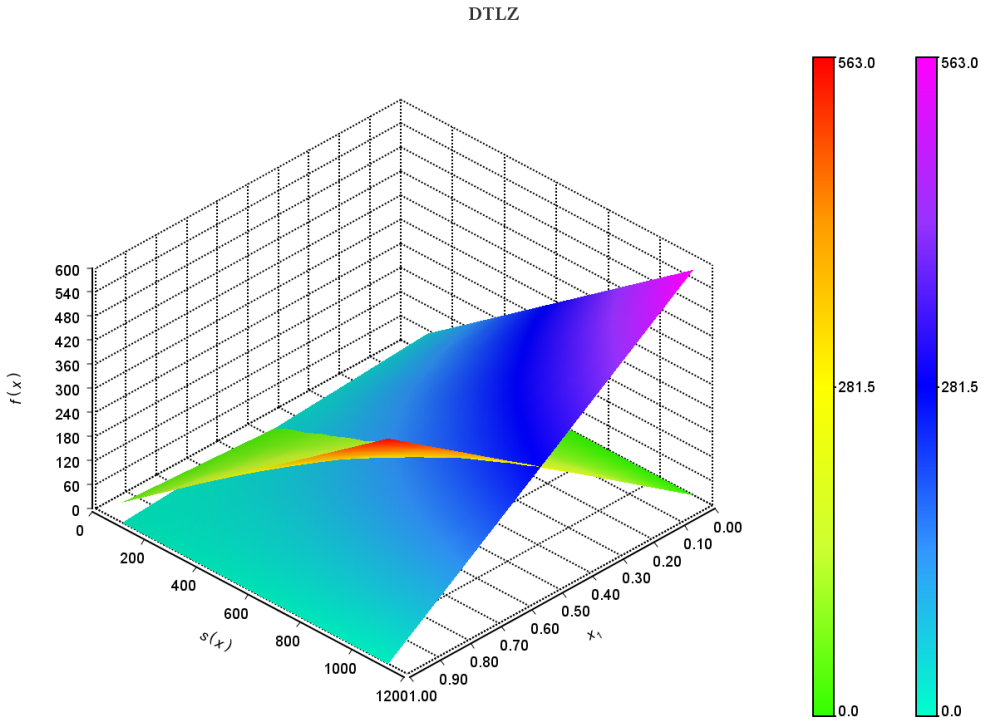


Abb. 5.2: Testfunktion DTLZ

5.1.2 Technische Aufgabe

Für das technische Problem wurde eine Aufgabe aus dem Bereich der Getriebeoptimierung, speziell Planetengetriebeoptimierung, gewählt. Die Planetengetriebeoptimierung ist eine konkrete Anwendung der diskreten Optimierung, da ein Planetengetriebe aus mehreren Zahnrädern mit einer diskreten Anzahl an Zähnen besteht, welche die Parameter der Optimierungsaufgabe darstellen. Ein spezielles Planetengetriebe ohne Zwischenrad und Stufenplanet, das so genannte *Minusgetriebe*, ist in Abb. 5.3, welche aus [33] entnommen wurde, abgebildet.
Die Optimierungsaufgabe wurde im Rahmen der Bachelorarbeit mit Hilfe der VDI-Richtlinie 2157 [34] aufgestellt. Da diese Aufgabe neu entwickelt wurde, sind keine Referenzlösungen, wie für die analytischen Aufgaben, bekannt. Zur Veranschaulichung der eingeführten Parameter, Zielfunktionen und Nebenbedingungen werden diese für das Planetengetriebe aus Abb. 5.3 in einem durchgängigen Beispiel bestimmt.

Das Ziel bei der Konstruktion eines Planetengetriebes besteht darin, für die Planetenräder (kurz Planeten), das Sonnenrad (kurz Sonne) und das Hohlrad geeignete Zahnräder zu finden, die eine vorgegebene Standübersetzung aufbringen und bestimmten Einbaubedingungen genügen.
Für die Modellierung der Aufgabe benötigt man zunächst die Variablen mit denen man die Zielfunktionen und Nebenbedingungen definieren kann. Bei dieser Aufgabe sind die Stellparameter die Zähnezahlen der Sonne z_S, des Hohlrades z_H und der Planeten z_P sowie die Anzahl der einge-

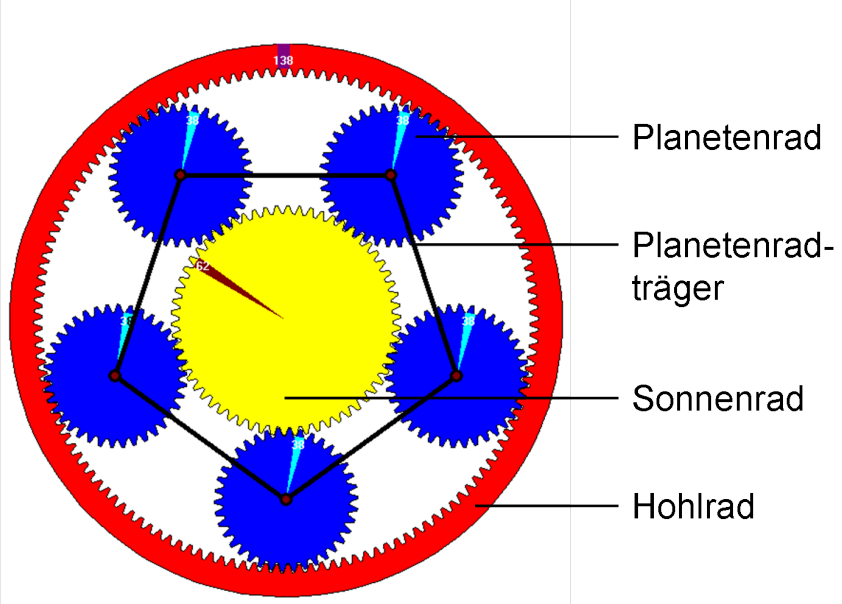


Abb. 5.3: Minusgetriebe mit fünf Planeten

bauten Planeten p. Dabei ist zu beachten, dass es sich bei dem Hohlrad um ein innenverzahntes Zahnrad handelt, für das die Zähnezahl immer negativ ist.

Beispiel 5.1:
Für das in Abb. 5.3 dargestellte Planetengetriebe haben die Parameter die Werte $z_S = 62$, $z_H = -138$, $z_P = 38$ und $p = 5$.

Die Standübersetzung eines Planetengetriebes ist das Verhältnis zwischen der Zähnezahl des Hohlrades und der Sonne $i = \frac{z_H}{z_S}$. Das erste Ziel ist es nun, die Abweichung der Standübersetzung i von einem vorgegebenen Wert i_0 zu minimieren. Unter Verwendung der quadratischen Abweichung ergibt sich die erste Zielfunktion:

$$f_1(z_S, z_H) = \left(\frac{z_H}{z_S} - i_0\right)^2 \rightarrow min \tag{5.3}$$

Das zweite Ziel besteht darin, das Verhältnis der Zähnezahl zwischen Sonne und Planeten bzw. Planeten und Hohlrad möglichst "krumm" zu gestalten, um eine einseitige Abnutzung der Zahnräder zu verhindern. Dazu wird der gebrochene Anteil dieser Verhältnisse maximiert:

$$\begin{aligned} f_{2_1}(z_S, z_P) &= \frac{z_S}{z_P} - \left\lfloor \frac{z_S}{z_P} \right\rfloor \rightarrow max \\ f_{2_2}(z_P, z_H) &= \frac{z_P}{z_H} - \left\lfloor \frac{z_P}{z_H} \right\rfloor \rightarrow max \end{aligned} \tag{5.4}$$

Da das Hauptaugenmerk jedoch auf der ersten Zielfunktion liegt, werden die beiden Funktionen f_{2_1} und f_{2_2} nicht einzeln maximiert, sondern addiert und anschließend die Summe maximiert:

$$f_2(z_S, z_H, z_P) = f_{2_1}(z_S, z_P) + f_{2_2}(z_P, z_H) = \frac{z_S}{z_P} - \left\lfloor \frac{z_S}{z_P} \right\rfloor + \frac{z_P}{z_H} - \left\lfloor \frac{z_P}{z_H} \right\rfloor \rightarrow max \tag{5.5}$$

Beispiel 5.1: Fortsetzung

Die Standübersetzung des Beispiel-Planetengetriebes ergibt sich zu

$$i = \frac{z_H}{z_S} = \frac{-138}{62} \approx -2.226$$

Das Verhältnis zwischen Sonne und Planeten beträgt $\frac{z_S}{z_P} = \frac{62}{38} \approx 1.6316$, das Verhältnis zwischen Planeten und Hohlrad hat den Wert $\frac{z_P}{z_H} = \frac{38}{-138} \approx -0.2754$.

Eine Nebenbedingung ergibt sich aus der Einbaubedingung des Minusgetriebes, welche die gleichmäßige Verteilung der Planeten am Umfang der Sonne sowie des Hohlrades sicherstellt. Sie kann wie folgt formuliert werden:

$$\frac{z_S - z_H}{p} = k \tag{5.6}$$

wobei k ganzzahlig sein muss. Um diese Bedingung in die Optimierungsaufgabe aufzunehmen, wird nicht die exakte Ganzzahligkeit gefordert, da diese aufgrund der Rechnerungenauigkeit nicht gewährleistet werden kann. Der gebrochene Anteil von k soll stattdessen kleiner als eine vorgegebene Schranke, z.B. 0.1, sein. Es ergibt sich somit die folgende Nebenbedingung:

$$g_1(z_S, z_H, p) = \frac{z_S - z_H}{p} - \left\lfloor \frac{z_S - z_H}{p} \right\rfloor < 0.1 \tag{5.7}$$

Betrachtet man das Planetengetriebe aus Abb. 5.3, so erkennt man, dass der Durchmesser d_P eines Planeten halb so groß sein muss wie die Differenz zwischen Hohlrad-Durchmesser $-d_H$[1] und Sonnen-Durchmesser d_S, d.h. es gilt

$$d_P = \frac{-d_H - d_S}{2} \tag{5.8}$$

Die Zähnezahl eines Zahnrades kann aus dessen Durchmesser mit Hilfe des Moduls m berechnet werden. Der Modul ist das Verhältnis zwischen Durchmesser und Zähnezahl eines Zahnrades und gibt an, wie die Zähne am Zahnradumfang verteilt sind. Da die Zahnräder des Planetengetriebes ineinander greifen, muss diese Verteilung und damit der Modul für alle Zahnräder gleich sein. Für die Zähnezahl ergibt sich somit:

$$z_S = \frac{d_S}{m},\ z_H = \frac{d_H}{m},\ z_P = \frac{d_P}{m} \tag{5.9}$$

Setzt man (5.9) in Gleichung (5.8) ein, so erhält man:

$$z_P * m = \frac{-z_H * m - z_S * m}{2}$$

Die Zähnezahl der Planeten lässt sich somit über die Zähnezahl von Sonne und Hohlrad mit Hilfe der Formel

$$z_P = -\frac{z_H + z_S}{2} \tag{5.10}$$

berechnen. Dieser Zusammenhang ist, wieder unter Berücksichtigung der Rechnerungenauigkeit, in der zweiten Nebenbedingung der OA festgehalten:

$$g_2(z_S, z_H, z_P) = \left| z_P + \frac{z_H + z_S}{2} \right| < 0.4 \tag{5.11}$$

[1] Die Negativität des Durchmessers ergibt sich wieder aus der Innenverzahnung des Hohlrades.

Beispiel 5.1: Fortsetzung

Für das Planetengetriebe ergibt sich für Gleichung (5.6)

$$\frac{z_S - z_H}{p} = \frac{62 - (-138)}{5} = 40$$

Da 40 eine ganze Zahl ist, ist die Einbaubedingung erfüllt.

Gleichung (5.10) ergibt sich zu

$$\begin{aligned} z_P &= -\frac{z_H + z_S}{2} \\ 38 &= -\frac{(-138) + 62}{2} \\ 38 &= 38 \end{aligned}$$

Die zweite Nebenbedingung ist somit ebenfalls erfüllt.

Für die Box-Restriktionen der einzelnen Zähnezahlen wurden aus der Praxis bekannte Werte für Zahnräder benutzt. Dabei wurde die prinzipielle Einbaumöglichkeit der Zahnräder zunächst außer Acht gelassen, da diese durch die Nebenbedingungen geregelt wird. Die Anzahl der Planeten wurde auf 3 bis 6 eingeschränkt.

Mit den Zielfunktionen (5.3) und (5.5), den Nebenbedingungen (5.7) und (5.11) sowie den Box-Restriktionen kann nun die Optimierungsaufgabe zusammenfassend formuliert werden. Für die Standübersetzung i_0 wurde außerdem der Wert -1.67 vorgegeben.

$$\begin{aligned} f_1(z_S, z_H) &= \left(\frac{z_H}{z_S} + 1.67\right)^2 \rightarrow min \\ f_2(z_S, z_H, z_P) &= \frac{z_S}{z_P} - \left\lfloor \frac{z_S}{z_P} \right\rfloor + \frac{z_P}{z_H} - \left\lfloor \frac{z_P}{z_H} \right\rfloor \rightarrow max \\ g_1(z_S, z_H, p) &= \frac{z_S - z_H}{p} - \left\lfloor \frac{z_S - z_H}{p} \right\rfloor < 0.1 \\ g_2(z_S, z_H, z_P) &= \left| z_P + \frac{z_H + z_S}{2} \right| < 0.4 \\ z_S &\in \{17, 18, ..., 76\} \\ z_H &\in \{-113, -112, ..., -17\} \\ z_P &\in \{17, 18, ..., 80\} \\ p &\in \{3, 4, 5, 6\} \end{aligned} \tag{5.12}$$

5.2 Vergleich und Auswertung

Für die Auswertung der Testaufgaben wurden diese mit unterschiedlicher Diskretisierung von jedem der vier Optimierungsverfahren zehnmal gelöst. Das Nelder-Mead-Verfahren und die Partikelschwarmoptimierung wurden dabei jeweils zehnmal mit und zehnmal ohne Intervall-Substitution ausgeführt. Die Parametereinstellungen für die Verfahren sind in Anhang B festgehalten.

5.2.1 Analytische Aufgaben

Für die Diskretisierung der Aufgabe Himmelblau wurden die in Tabelle 5.1 aufgeführten Schrittweiten genutzt. Die Schrittweite 0 bedeutet dabei, dass der Parameter nicht diskretisiert wurde.

x_1	0.8	1	0.5	0.4	1.2	2.4	2	2.5	2.5	3.5	3.5	0	0.7	0.7
x_2	0.8	0.5	1	0.3	2.4	1.2	3	2.5	3.5	2.5	3.5	0.7	0	0.7

Tab. 5.1: Diskretisierung mit Schrittweite für Testfunktion Himmelblau

Außerdem wurde eine Diskretisierung mit zwei verschiedenen Mengen $M_1 = \{-5, -3, 4, 6\}$ und $M_2 = \{-4.7, -3.2, 0, 1.2, 5.8\}$ wie in Tabelle 5.2 dargestellt durchgeführt.

x_1	M_1	0	M_1	M_2	0	M_2	M_1	M_2
x_2	0	M_1	M_1	0	M_2	M_2	M_2	M_1

Tab. 5.2: Diskretisierung mit diskreter Menge für Testfunktion Himmelblau

Für die Testaufgabe DTLZ wurden insgesamt 14 Diskretisierungen mit Schrittweite und sechs Diskretisierungen mit vorgegebener Menge durchgeführt, welche in Tabelle 5.3 und 5.4 aufgeführt sind. Dabei besteht die Menge M_3 in Tabelle 5.4 aus den Werten $\{0, 0.4, 0.9\}$ und M_4 besitzt die Werte $\{0.45, 0.55, 0.9\}$.

x_1	0.01	0.1	0.2	0.25	0.3	0.35	0.4	0.5	0.55	0.6	0.7	0.8	0.9	1
x_2	0.01	0.1	0.2	0.25	0.3	0.35	0.4	0.5	0.55	0.6	0.7	0.8	0.9	1
x_3	0.01	0.1	0.2	0.25	0.3	0.35	0.4	0.5	0.55	0.6	0.7	0.8	0.9	1
x_4	0.01	0.1	0.2	0.25	0.3	0.35	0.4	0.5	0.55	0.6	0.7	0.8	0.9	1
x_5	0.01	0.1	0.2	0.25	0.3	0.35	0.4	0.5	0.55	0.6	0.7	0.8	0.9	1
x_6	0.01	0.1	0.2	0.25	0.3	0.35	0.4	0.5	0.55	0.6	0.7	0.8	0.9	1

Tab. 5.3: Diskretisierung mit Schrittweite für Testfunktion DTLZ

x_1	M_3	M_3	M_3	M_4	M_4	M_4
x_2	M_3	M_3	M_4	M_4	M_4	M_3
x_3	M_3	M_3	M_4	M_4	M_4	M_3
x_4	M_3	M_4	M_4	M_4	M_3	M_3
x_5	M_3	M_4	M_4	M_4	M_3	M_3
x_6	M_3	M_4	M_4	M_4	M_3	M_3

Tab. 5.4: Diskretisierung mit diskreter Menge für Testfunktion DTLZ

Ein erstes Kriterium für die Bewertung der Optimierungsverfahren ist die exakte Abbildung der Diskretheitsforderung der Testaufgabe. So wurde für die vier Verfahren in fast allen Fällen die

Diskretisierung exakt eingehalten. Jedoch konnten die Verfahren Nelder-Mead und PSO bei einer Intervall-Substitution in der Aufgabe DTLZ die Randpunkte der Pareto-Menge nicht immer exakt finden, wodurch theoretisch unzulässige Punkte ermittelt wurden. Dies liegt in der Wahl der Tranfsformationsvorschrift begründet. Da in einem Rechner die Grenzen $\pm\infty$ nicht exakt dargestellt werden können, werden stattdessen konkrete Zahlen benutzt. Dadurch ergeben sich besonders an den Intervallgrenzen Ungenauigkeiten bei der Rücktransformation der Werte aus dem Intervall $(-\infty, \infty)$ in das ursprüngliche Intervall $[x_L, x_U]$.
Es lässt sich also sagen, dass die Diskretisierung, mit wenigen Abstrichen bei der Intervall-Substitution, für alle vier Optimierungsverfahren gelungen ist.

Nun muss noch überprüft werden, ob die von den Optimierungsverfahren ermittelten Lösungen auch den optimalen Lösungen entsprechen. Dafür wurde bei der einkriteriellen Aufgabe Himmelblau für jedes Verfahren die häufigste der Lösungen, die in den zehn Durchläufen berechnet wurden, ermittelt und mit der optimalen Lösung verglichen. Ein Verfahren kann dann als gut eingeschätzt werden, wenn die gefundene Lösung dem Optimum entspricht oder der Zielfunktionswert nur wenig vom Optimum abweicht. Für die multikriterielle Testaufgabe DTLZ ist die Anzahl und die Genauigkeit der paretooptimalen Punkte ausschlaggebend. So wurde die Anzahl der gefundenen optimalen Punkte über alle Durchläufe gemittelt und gleichzeitig die Exaktheit der Punkte beachtet.
Das Verfahren EMMOA schnitt mit Abstand am besten bei der Auswertung ab. So wurde für die Testaufgabe Himmelblau mit allen Diskretisierungen die optimale Lösung exakt bestimmt. Auch bei der Aufgabe DTLZ wurde die Pareto-Menge meist exakt approximiert. Lediglich für die Schrittweite 0.01 konnten nur wenige paretooptimale Punkte gefunden werden.
Die Lösungen der Partikelschwarmoptimierung und des DiRect-Verfahrens waren in den meisten Fällen ebenfalls exakt. Für die wenigen Ergebnisse der Himmelblau-Aufgabe, bei denen die optimale Lösung nicht gefunden wurde, waren die Funktionswerte der ermittelten Lösungen nur wenig größer als das Minimum. Auch die Anzahl der gefundenen paretooptimalen Punkte für die Aufgabe DTLZ entsprach meistens der Mächtigkeit der exakten Pareto-Menge. Ein Problem ergab sich bei dem Verfahren Dividing-Rectangles für eine große Schrittweite. So wurde für die Aufgabe DTLZ bei einer Schrittweite größer als 0.6 nur ein zulässiger Punkt gefunden, welcher dann als paretooptimaler Wert ausgegeben wurde. Dieses Verhalten lässt sich dadurch erklären, dass durch die Diskretisierung die Suche nach neuen Mittelpunkten immer wieder auf den bereits ermittelten Startpunkt führt. Somit wird während des ganzen Algorithmus nur ein Wert berechnet und es können keine weiteren Punkte gefunden werden.
Das Nelder-Mead-Verfahren hatte die meisten Schwierigkeiten bei der Berechnung optimaler Lösungen. Für die Himmelblau-Aufgabe wurden nur selten die optimalen Werte ermittelt und die gefundenen Lösungen lagen oft weit von den Optimalwerten entfernt. Auch für die Testaufgabe DTLZ wurden oft falsche Werte für die Pareto-Menge ermittelt.

5.2.2 Technische Aufgabe

Die technische Aufgabe "Planetengetriebe" wurde von den vier Optimierungsverfahren jeweils zehnmal gelöst, wobei die Partikelschwarmoptimierung zehnmal mit und zehnmal ohne Intervall-Substitution durchgeführt wurde. Das Nelder-Mead-Verfahren hingegen konnte nur ohne Intervall-Substitution ausgewertet werden, da bei einer Durchführung mit Intervall-Substitution keine zulässigen Punkte gefunden werden konnten. Anschließend wurden alle Lösungen, die von den Optimierungsverfahren berechnet wurden, zusammengefasst und die paretooptimalen Lösungen ermittelt, welche in Tabelle 5.5 aufgeführt sind.

z_S	z_H	z_P	p	f_1	f_2
60	-100	20	5	1.1111E-05	0.8
51	-85	17	4	1.1111E-05	0.8
63	-105	21	3	1.1111E-05	0.8
57	-95	19	4	1.1111E-05	0.8
54	-90	18	6	1.1111E-05	0.8
54	-90	18	3	1.1111E-05	0.8
65	-109	22	6	4.7929E-05	1.7527
67	-101	17	3	0.0264	1.7729
68	-102	17	5	0.0289	0.8333

Tab. 5.5: Paretooptimale Lösungen der technischen Aufgabe "Planetengetriebe"

Es sei angemerkt, dass die ersten sechs Lösungen zwar mathematisch gut, technisch aber eher unbrauchbar sind, da die zweite Zielfunktion nicht "krumm" genug ist (vgl. Aufstellung der zweiten Zielfunktion in 5.1.2).

Die Verfahren können nun hinsichtlich der Anzahl der gefundenen paretooptimalen Lösungen verglichen werden. Die Optimierungserfahren EMMOA und PSO ohne Intervall-Substitution fanden dabei die meisten Punkte der Pareto-Menge. Das DiRect- und das Nelder-Mead-Verfahren ermittelten hingegen keinen dieser Punkte, aber die Ergebnisse, die von dem Verfahren DiRect berechnet wurden, lagen fast alle in der Nähe der Pareto-Front und waren technisch brauchbar, d.h. die Abweichung von der gegebenen Standübersetzung war gering und die Verhältnisse der Zähnezahlen waren "krumm". Das Nelder-Mead-Verfahren hingegen lieferte Lösungen, die teilweise weit entfernt von der Pareto-Front lagen und daher eine große Abweichung von der gegebenen Standübersetzung hatten. Für die Partikelschwarmoptimierung mit Intervall-Substitution trat indes wieder das Problem auf, dass die zulässigen diskreten Werte nicht exakt berechnet wurden, so erhielt man z.B. für die Zähnezahl eines Planetenrades den Wert 17.00001927 und für die Anzahl der Planeten 5.999999.

5.2.3 Zusammenfassung

Aus den in den vorangegangenen Abschnitten gewonnenen Erkenntnissen lässt sich folgern, dass alle vier Optimierungsverfahren diskretisiert werden konnten. Zur Lösung einer diskreten Optimierungsaufgabe empfiehlt sich dabei das Verfahren EMMOA, insbesondere, wenn es sich um eine Aufgabe handelt, bei der die Zielfunktionsauswertungen zeitaufwändig sind. Für Optimierungsaufgaben, deren Zielfunktionen schnell auszuwerten sind, können auch die Verfahren DiRect und PSO eingesetzt werden, da bei diesen Verfahren der zeitliche Aufwand zur Lösung solcher Aufgaben erheblich geringer ist als bei EMMOA und die ermittelten Lösungen ebenfalls gut sind. Jedoch sollte bei der Partikelschwarmoptimierung auf eine Intervall-Substitution aufgrund der genannten Ungenauigkeiten verzichtet werden. Das Nelder-Mead-Verfahren eignet sich hingegen nicht zur Lösung diskreter Optimierungsprobleme.

6 Zusammenfassung und Ausblick

Das Ziel dieser Arbeit bestand darin, ausgewählte Optimierungsverfahren der ein- und multikriteriellen Optimierung zu diskretisieren. Dazu wurde zunächst im 2. Kapitel ein Überblick über die Optimierung gegeben, bei der sowohl einkriterielle, multikriterielle als auch diskrete Optimierungsaufgaben betrachtet wurden. Im Anschluss erfolgte in Kapitel 3 eine Beschreibung der vier ausgewählten Optimierungsverfahren, deren Diskretisierung in Kapitel 4 erläutert wurde. Das letzte Kapitel beschäftigte sich mit der Bewertung der diskretisierten Verfahren. Es wurden zwei analytische und eine technische Aufgabe vorgestellt und mit deren Hilfe eine Einschätzung der Optimierungsverfahren gegeben.

Es wurde in dieser Arbeit gezeigt, dass die vier Optimierungsverfahren *Dividing-Rectangles* und *Nelder-Mead*, beides deterministische Verfahren, sowie die *Partikelschwarmoptimierung* und der *Enhanced Modelbased Multiobjective Optimization Algorithm*, zwei stochastische Methoden, diskretisiert werden können.
Bei der Auswertung der Testaufgaben wurde festgestellt, dass das Nelder-Mead-Verfahren nicht für eine Diskretisierung geeignet ist, da es zu oft schlechte Resultate lieferte. Auch das DiRect-Verfahren ist nur bedingt für diskrete Optimierungsaufgaben einsetzbar, denn für eine grobe Diskretisierung werden durch den Algorithmus insgesamt zu wenige zulässige Lösungen gefunden und es kann daher kaum von einer Optimierung gesprochen werden. Für eine feinere Diskretisierung liefert das DiRect-Verfahren jedoch gute Ergebnisse. Auch für die Partikelschwarmoptimierung und das Verfahren EMMOA konnte gezeigt werden, dass bei einer beliebigen Diskretisierung zufriedenstellende Resultate erreicht werden können.

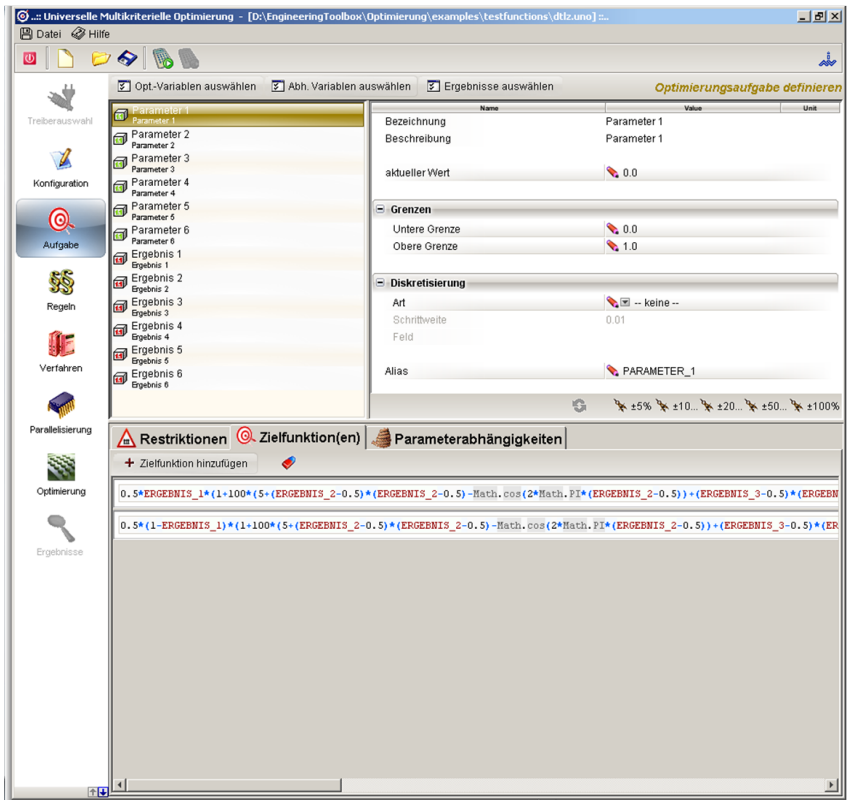
Da im Rahmen dieser Bachelorarbeit nur die Möglichkeit bestand, einen Teil der in der *EngineeringToolbox* vorhandenen Optimierungsverfahren zu diskretisieren, besteht eine weiterführende Aufgabe zu diesem Thema darin, weitere Verfahren zu diskretisieren. Dazu können die aus Kapitel 4 gewonnenen Erkenntnisse zur Diskretisierung der dort aufgeführten Verfahren genutzt werden.
Zusätzlich zur Diskretisierung der vorhandenen Optimierungsverfahren wäre auch die Implementierung von Verfahren, welche speziell für diskrete Optimierungsprobleme entwickelt wurden, überlegenswert. In Frage kämen dafür z.B. die Branch-and-Bound-Methode oder das Branch-and-Cut-Verfahren, welche in Kapitel 2.4.1 beschrieben wurden.

Weiterhin sollte bei einer Diskretisierung mit Schrittweite und gleichzeitiger Intervall-Substitution des Wertebereichs eines Parameters, wie z.B. beim Nelder-Mead-Verfahren oder der Partikelschwarmoptimierung, eine alternative Lösung für die Initialisierung der Klasse *CDiscreteHandler* als die derzeitig implementierte Variante gefunden werden, da durch das genannte Speicherplatzproblem Schwierigkeiten bei der Durchführung einer diskreten Optimierung entstehen können.

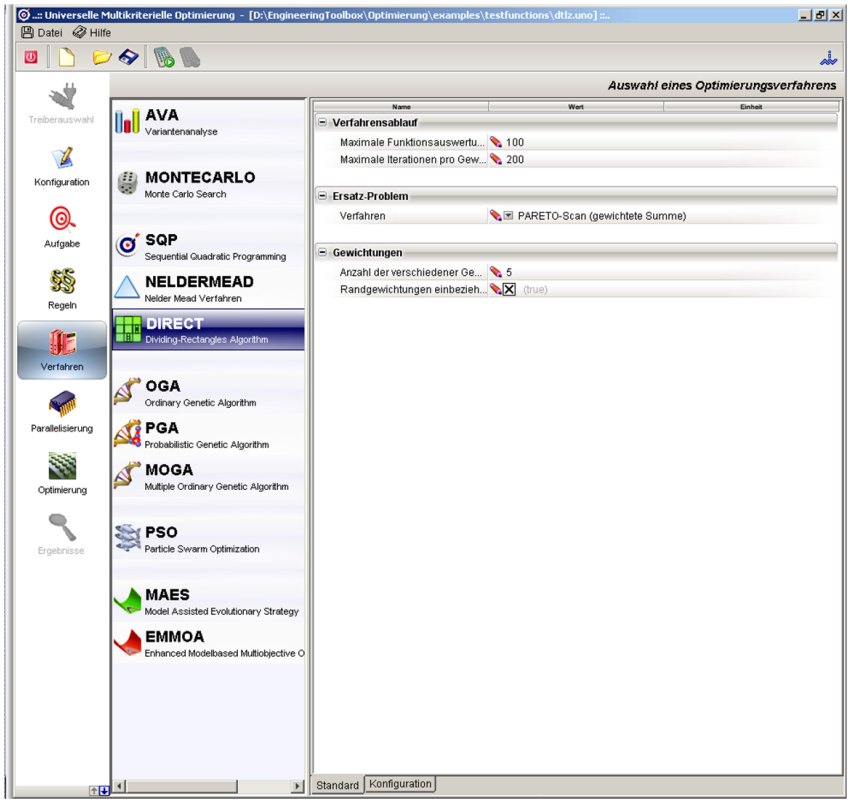
Anhang

A EngineeringToolbox

Die *EngineeringToolbox* der IAV GmbH ist ein Softwareprogramm, welches u.a. zur Toleranzsimulation, Optimierung sowie zur Aufbereitung und Visualisierung von Daten genutzt werden kann. Das Optimierungswerkzeug UNO (Universal Optimizer) aus der *EngineeringToolbox* dient der Lösung verschiedener OA und wird im Folgenden kurz vorgestellt.
Die Oberfläche von UNO gliedert sich in mehrere Teile. Um eine Optimierung durchführen zu können, müssen zuerst über zwei Schaltflächen verschiedene Voreinstellungen zur Treiberauswahl und Konfiguration getroffen werden, danach kann die OA formuliert werden. Wie in Abb. A.1a zu sehen ist, werden dafür zunächst die Parameter mit ihren Grenzen und einem Startwert sowie einer möglichen Diskretisierung definiert. Anschließend werden die Zielfunktionen, Restriktionen und eventuelle Parameterabhängigkeiten aufgestellt. Außerdem können über eine weitere Schaltfläche bestimmte Regeln, die sich aus technischen Gegebenheiten ergeben, festgelegt werden. Danach folgt die Auswahl eines Optimierungsverfahrens, wie in Abb. A.1b dargestellt, und dessen Konfiguration. So können z.B. Abbruchbedingungen, Toleranzschranken oder andere Verfahrensparameter festgelegt werden. Nach der Auswahl des Verfahrens kann die Optimierung gestartet werden. Während der Berechnung werden die gefundenen Lösungen in einem 2D-Diagramm angezeigt. Außerdem wird die Anzahl der zulässigen und unzulässigen Lösungen sowie eine Hochrechnung der verbleibenden Zeit ausgegeben (vgl. Abb. A.1c). Ist die Optimierung beendet, wird automatisch zur Ergebnisausgabe gewechselt. In dieser werden die Anzahl der optimalen Lösungen und die einzelnen Ergebnisse sowohl in einer Tabelle als auch graphisch angezeigt (siehe Abb. A.1d).

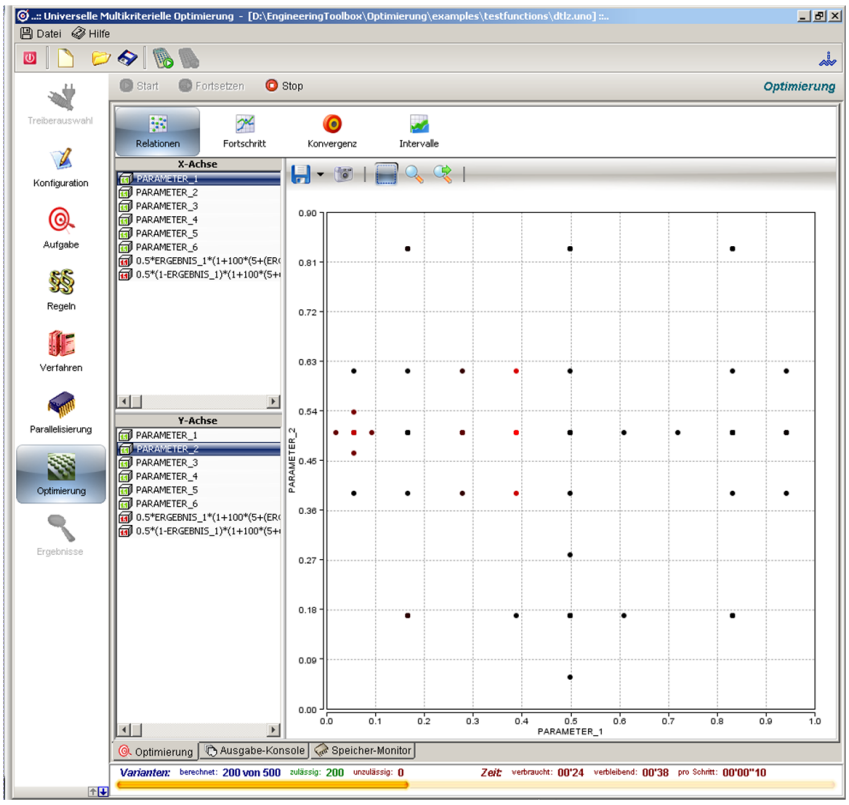


(a) Definition der Aufgabe

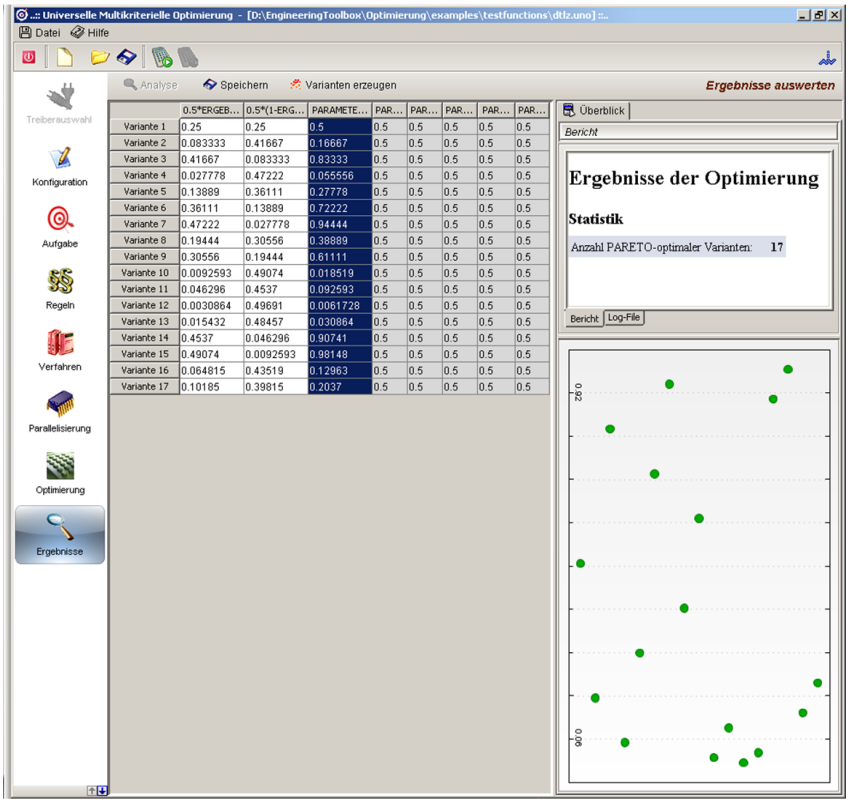


(b) Auswahl des Optimierungsverfahrens

Abb. A.1: Oberfläche des Optimierungswerkzeuges UNO



(c) Optimierungsverlauf



	0.5*ERGEB...	0.5*(1-ERG...	PARAMETE...	PAR...	PAR...	PAR...	PAR...	PAR...
Variante 1	0.25	0.25	0.5	0.5	0.5	0.5	0.5	0.5
Variante 2	0.083333	0.41667	0.16667	0.5	0.5	0.5	0.5	0.5
Variante 3	0.41667	0.083333	0.83333	0.5	0.5	0.5	0.5	0.5
Variante 4	0.027778	0.47222	0.055556	0.5	0.5	0.5	0.5	0.5
Variante 5	0.13889	0.36111	0.27778	0.5	0.5	0.5	0.5	0.5
Variante 6	0.36111	0.13889	0.72222	0.5	0.5	0.5	0.5	0.5
Variante 7	0.47222	0.027778	0.94444	0.5	0.5	0.5	0.5	0.5
Variante 8	0.19444	0.30556	0.38889	0.5	0.5	0.5	0.5	0.5
Variante 9	0.30556	0.19444	0.61111	0.5	0.5	0.5	0.5	0.5
Variante 10	0.0092593	0.49074	0.018519	0.5	0.5	0.5	0.5	0.5
Variante 11	0.046296	0.4537	0.092593	0.5	0.5	0.5	0.5	0.5
Variante 12	0.0030864	0.49691	0.0061728	0.5	0.5	0.5	0.5	0.5
Variante 13	0.015432	0.48457	0.030864	0.5	0.5	0.5	0.5	0.5
Variante 14	0.4537	0.046296	0.90741	0.5	0.5	0.5	0.5	0.5
Variante 15	0.49074	0.0092593	0.98148	0.5	0.5	0.5	0.5	0.5
Variante 16	0.064815	0.43519	0.12963	0.5	0.5	0.5	0.5	0.5
Variante 17	0.10185	0.39815	0.2037	0.5	0.5	0.5	0.5	0.5

(d) Ergebnis der Aufgabe

Abb. A.1: Oberfläche des Optimierungswerkzeuges UNO

B Parameterauswahl

B.1 Dividing-Rectangles-Verfahren

Für alle Aufgaben wurde die maximale Anzahl der Teilungen auf 100 beschränkt. Außerdem durften höchstens 5 000 Zwischenlösungen gespeichert werden. Der Penalty-Wert für unzulässige Lösungen wurde auf 100 000 gesetzt. Die weiteren Einstellungen finden sich in der nachfolgenden Tabelle B.1.

DIRECT	Himmelblau	DTLZ	Planetengetriebe
Max. Funktionsauswertungen	1 000	500	1 000
Max. Iterationen (pro Gewicht)	200	200	200
Ersatz-Problem	–	Pareto-Scan gewichtete Summe	Pareto-Scan gewichtete Summe
Anzahl der Gewichte	–	10	5

Tab. B.1: Parametereinstellung für das Dividing-Rectangles-Verfahren

B.2 Nelder-Mead-Verfahren

Der Penalty-Wert für das Nelder-Mead-Verfahren wurde ebenfalls für alle Testaufgaben auf 100 000 festgelegt. Außerdem wurde der Algorithmus abgebrochen, wenn sich der Zielfunktionswert der neuen Lösung nicht mehr als 10^{-5} von den vorherigen Werten unterschied. Die Parametereinstellungen, die sich für die Aufgaben unterscheiden, sind in Tabelle B.2 aufgelistet.

NM	Himmelblau	DTLZ	Planetengetriebe
Max. Iterationen (pro Gewicht)	50	20	50
Anzahl Startlösungen (pro Gewicht)	10	10	5
Ersatz-Problem	–	Pareto-Scan gewichtete Summe	Pareto-Scan gewichtete Summe
Anzahl der Gewichte	–	20	5

Tab. B.2: Parametereinstellung für das Nelder-Mead-Verfahren

B.3 Partikelschwarmoptimierung

Bei allen drei Testaufgaben rechnete der Algorithmus der Partikelschwarmoptimierung mit einer festen Nachbarschaft der Größe 5. Außerdem wurden die Konstanten zur Berechnung der neuen Geschwindigkeit (vgl. Gleichung (3.15)) auf die folgenden Werte gesetzt:

$$w \in [0.1, 1],\ c_1 = c_2 = 2$$

Der Algorithmus wurde abgebrochen, wenn eine Genauigkeit von 10^{-5} für den Funktionswert erreicht wurde. Weitere Einstellungen für die Partikelschwarmoptimierung sind in Tabelle B.3 zu finden.

PSO	Himmelblau	DTLZ	Planetengetriebe
Anzahl Partikel	20	30	25
Anzahl Generationen	20	50	50

Tab. B.3: Parametereinstellung für die Partikelschwarmoptimierung

B.4 Enhanced Modelbased Multiobjective Optimization Algorithm

Für jede der drei Aufgaben wurde als Auffüllkriterium das empfohlene *Multi Criteria Sampling* (siehe [28]) genutzt. Außerdem wurden bei der Initialisierung der Startwerte auch unzulässige Werte zugelassen, da sich sonst kein Modell aufbauen ließ. In der nachfolgenden Tabelle sind die weiteren Parametereinstellungen aufgeführt.

EMMOA	Himmelblau	DTLZ	Planetengetriebe
Anzahl zu berechnender Punkte	100	100	50
Punkte für Modellbildung	50	50	30
Anzahl Startpunkte	21	65	43
Populationsgröße für Tiefensuche	30	30	50
Generationenanzahl für Tiefensuche	50	50	30
Populationsgröße für Breitensuche	30	30	50
Generationenanzahl für Breitensuche	50	50	30

Tab. B.4: Parametereinstellung für EMMOA

Literaturverzeichnis

[1] Benker, Hans: *Mathematische Optimierung mit Computeralgebrasystemen*. Springer-Verlag, 2003.

[2] Biesenbach, Andreas: *Multi-Site-Scheduling in der chemischen Industrie*. Deutscher Universitäts-Verlag | GWV Fachverlage GmbH, 2007.

[3] Björkman, Mattias und Holmström, Kenneth: *Global Optimization Using the DIRECT Algorithm in Matlab*. Advanced Modeling and Optimization (AMO), S. 17–37, 1999.

[4] Borgelt, Christian: *Ameisenkoloniealgorithmen und andere schwarmbasierte Optimierungsverfahren*. Vortrag, Otto-von-Guericke-Universität Magdeburg, Dezember 2005.

[5] Byatt, David: *Convergent Variants of the Nelder-Mead-Algorithm*. Masterarbeit, University of Canterbury, 2000.

[6] Deb, Kalyanmoy: *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, 2. Auflage, 2002.

[7] Dempe, Stephan und Schreier, Heiner: *Operations Research: Deterministische Modelle und Methoden*. B. G. Teubner Verlag | GWV Fachverlage GmbH, 2006.

[8] Finkel, Daniel E.: *DIRECT Optimization Algorithm User Guide*. Center for Research in Scientific Computation. North Carolina State University, März 2003.

[9] Gablonsky, Jörg M.: *Modifications of the Direct Algorithm*. Dissertation, North Carolina State University Raleigh, April 2001.

[10] Geiger, Carl und Kanzow, Christian: *Ergänzungen zu dem Buch: Numerische Verfahren zur Lösung unrestringierter Optimierungsaufgaben*. Springer-Verlag, 1999.

[11] Geiger, Carl und Kanzow, Christian: *Theorie und Numerik restringierter Optimierungsaufgaben*. Springer-Verlag, 2002.

[12] Göpfert, Alfred; Bittner, Leonhard; Elster, Karl-Heinz; Nožička, František; Piehler, Joachim und Tichatschke, Rainer (Hrsg.): *Lexikon der Optimierung*. Akademie-Verlag Berlin, 1986.

[13] Gumpert, Werner: *Optimierung mit dem PC*. VDI Verlag GmbH, 1996.

[14] Handl, Julia; Lovell, Simon C. und Knowles, Joshua: *Multiobjectivization by Decomposition of Scalar Cost Functions*. Parallel Problem Solving from Nature (PPSN X), S. 31–40, 2008.

[15] Ingenieurgesellschaft Auto und Verkehr: *IAV Firmenprofil*. URL: http://www.iav-inside.com/de/unternehmen/profil.php, Stand: 23.08.2010.

[16] Jones, Donald R.; Perttunen, Cary D. und Stuckman, Bruce E.: *Lipschitzian Optimization Without the Lipschitz Constant*. Journal of Optimization Theory and Application, S. 157–181, 1993.

[17] Knowles, Joshua D.; Watson, Richard A. und Corne, David W.: *Reducing Local Optima in Single-Objective Problems by Multi-objectivization*. Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization (EMO 2001), S. 269–283, 2001.

[18] Korbut, Aleksandr A. und Finkelstein, Julij J.: *Diskrete Optimierung*. Akademie-Verlag Berlin, 1971.

[19] Lammert, Daniel: *Direct Search Algorithms*. Vortrag, Universität zu Köln, Dezember 2008.

[20] Li, Lily D; Li, Xiaodong und Yu, Xinghuo: *Power Generation Loading Optimization using a Multi-Objective Constraint-Handling Method via PSO Algorithm*. The IEEE International Conference on Industrial Informatics (INDIN 2008), S. 1632–1637, 2008.

[21] Meyer, Dany: *Modellbasierte Mehrzieloptimierung mit Neuronalen Netzen und Evolutionsstrategien*. Dissertation, Technische Universität Ilmenau, Oktober 2003.

[22] Miettien, Kaisa M.: *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, 4. Auflage, 2004.

[23] Ratzlaff, Martin: *Untersuchung von Optimierungsverfahren der ein- und multikriteriellen Optimierung mit diskreten Parametern für Anwendungen in der Antriebsstrangauslegung*. Diplomarbeit, Hochschule für Technik, Wirtschaft und Kultur Leipzig, August 2007.

[24] Röber, Marcel: *Multikriterielle Optimierungsverfahren für rechenzeitintensive technische Aufgabenstellungen*. Diplomarbeit, Technische Universität Chemnitz, März 2010.

[25] Reichenbach, Michael: *Ranking 2010. Aufschwung nach der Krise*. ATZextra. Automotive Engineering Partners, S. 6–13, Mai 2010.

[26] Richter, Knut; Bachmann, Peter und Dempe, Stephan: *Diskrete Optimierungsmodelle*. VEB Verlag Technik Berlin, 1988.

[27] Rudolph, Günter und Preuss, Mike: *Ein Evolutionsverfahren zur Approximation äquivalenter Urbilder von Pareto-optimalen Zielvektoren*. Proceedings 18. Workshop Computational Intelligence, S. 30–39, 2008.

[28] Schmidt, Hansjörg: *Parallelisierung Ersatzmodell-gestützter Optimierungsverfahren*. Diplomarbeit, Technische Universität Chemnitz, Februar 2009.

[29] Schneider, Stefan: *Rechnergestützte, kooperativ arbeitende Optimierungsverfahren am Beispiel der Fabrikplanung*. Dissertation, Universität Gesamthochschule Kassel, März 2001.

[30] Stöcker, Martin: *Untersuchung von Optimierungsverfahren für rechenzeitaufwändige technische Anwendungen in der Motorenentwicklung*. Diplomarbeit, Technische Universität Chemnitz, Mai 2007.

[31] Stoica-Klüver, Christina; Klüver, Jürgen und Schmidt, Jörn: *Modellierung komplexer Prozesse durch naturanaloge Verfahren*. Vieweg+Teubner Verlag | GWV Fachverlage GmbH, 2009.

[32] Streng, Jürgen: *Optimierung und Analyse von Fachwerkstrukturen durch Neuronale Netze*. Dissertation, Universität Stuttgart, August 2001.

[33] Trempler, Uwe: *Kinematik ebener Planetengetriebe*. PC-Programm, 2006.

[34] Verein Deutscher Ingenieure: *VDI-Richtlinie 2157: Planetengetriebe*. Düsseldorf, Januar 2010.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Chemnitz, 4. September 2010